An Integrated Approach to General Software Monitoring

TR86-022

May 1986

Stephen Edward Duncan

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

The SoftLab Project

An Integrated Approach to General Software Monitoring

Stephen Edward Duncan

May 1986

Abstract

This thesis describes a system of general data collection and analysis tools for monitoring user programs and the Unix Kernel. The data is produced by sensors that are defined in a Sensor Descriptor Language, and can be placed in bother user and system code. The analysis tools treat the data as a relational database, with each sensor producing tuples in its own relation. Specific sensors have been installed in the file system source code and a sample database created to demonstrate the utility of the approach.

SoftLab Document No. 27

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27514

Acknowledgments

I would like to express my appreciation for the support and assistance of my thesis committee, Mahadev Satyanarayanan, John Smith, and most especially the committee chairman, Rick Snodgrass, in the development of the thesis document. Many helped in the project development, both in technical assistance and the contribution of ideas, Rick, Satya, Tim Seaver, and the teams at Carnegie-Mellon and Ohio State who suffered through early releases. Finally I would like to thank my wife, Lynne, for her support throughout the project.

Table Of Contents

Chapter 1

Introduction

The purpose of this master's project is to provide a general purpose method for instrumenting and monitoring software. Towards this end, a suite of general data collection and analysis tools for monitoring user programs and the UNIX kernel have been designed and implemented. Specific sensors have been installed in the file system source code, that have been used to generate a sample database used to demonstrate the utility of the tools. This tool suite will assist users with the evaluation of their software and will assist them in experiments performed on those systems.

The design of the monitoring system was driven by a set of questions concerning the Unix file system. These questions required a specific set of data retrieved via sensors in the Unix kernel, an accounting user program to control the sensors, and a monitoring system call to act as an interface between the accounting program and the sensors. A flexible method to investigate the data is provided in the set of analysis tools.

When the project was begun, only static observations of file systems existed [13] [18] [15]. To get a significant increase in the knowledge of file system usage dynamic information was needed. The original goal of this project was to provide this data. Implementing the software to record the dynamic use data showed that the approach could be modified from a specific monitoring system into a generalized monitoring system. During the course of this project, other researchers have since done some dynamic studies, notably Ousterhout [12] and Floyd [4], which will be discussed below with other monitoring approaches.

Following this introduction is a review of the current state of software monitoring and its relation to this project. Each of the remaining chapters discuss an aspect of the project's implementation. Chapter 3 discusses the data structures used in the system. Chapters 4, 5, and 6 discuss the data gathering parts of the system. The sensors are described in chapter 4 and in Appendix A, with the chapter covering the specific design and placement of the implemented sensors and the appendix covering the method of sensor generation. Chapter 5 details the requirements and workings of the monitoring system call, and chapter 6

discusses the accounting process with which it interacts. The tools to analyze the data are discussed in Chapter 7. The chapter discusses the paradigm used to view the data and gives a description of the high level implementation used to meet this paradigm. One appendix contains manual pages describing the user interface, while an additional appendix contains directions for installing a distribution of the system.

This document uses a set of fonts to distinguish types of special terms. *Italics* are used to introduce special terms and is used in displays of relational algebra expressions and C comments. C variables are presented in `typewriter` and keywords are in **`typewriter bold`**, while displays of terminal output are in **`typewriter bold`** and typed input is shown in `typewriter`. Unix file names and program names are presented in the body of the text in *helvetica slanted*.

Chapter 2

Software Monitoring

This chapter describes other work done in software monitoring as it relates to the approach taken in this project. This is organized into three sections. The first describes the tools available with the Unix operating system, the second describes work done by other researchers, and the third describes the approach taken here.

2.1. UNIX Tools

A small set of tools for software monitoring is supplied with the Unix operating system. These are summarized in the table below and discussed in the paragraphs that follow.

dbx	allows dynamic control and inspection of an executing process.
gprof/prof	provides a trace of the function calls made by a process.
iostat	reports device i/o statistics for the system.
vmstat	reports virtual memory statistics for the system.
ps	provides a report of all processes in the system.
tcov	performs instruction counting.
kernel tracing	provides information about certain events in the kernel.
print statments	inserted into code, provides flexible but primitive monitoring.

2.1.1. dbx

The purpose of *dbx* is to assist in tracing logic problems while developing a program. It can examine specifics of a program's execution and possesses facilities for tracing and recording data, but provides no information or control of system calls or of details within the operating system, and is limited to operating on a single process at a time. Programs that use *dbx* are compiled with an option that keeps additional data that are used in tracing the program.

2.1.2. gprof and prof

The related tools *gprof* [8] and *prof* are designed to show the control flow within a program. Both can be used on user processes; *gprof* can also be used on a specially configured kernel. Neither tool allows information other than execution of a function to be recorded, so that the state of the process can't be determined for a given function execution. This prevents the recovery of system call usage by a user from the information provided by the profiled kernel.

2.1.3. System statistics

Iostat and *vmstat* provide statistics on the operating system, while *ps* provides statistics on user processes. *Iostat* reports the i/o operations for devices and *vmstat* reports statistics on the operating system's virtual memory. Neither present any information regarding individual processes, only about the system as a whole. *Ps* provides information about processes, but only from the system process table, not from their internal state, such as the routine currently executing or the contents of variables.

2.1.4. kernel tracing

Tracing may be configured into the kernel. It provides for the recording of events in the kernel, and can be dynamically controlled. New events can't be easily added, nor can the information interrogated by the sensors be changed.

2.1.5. tcov (instruction counting)

tcov produces a statement-by-statement profile of a C program, useful for determining the often executed code sections and the determining test coverage. A program uses it by specifying an option to the compiler that does the actual work of inserting the monitoring code. Each time the program is run, the monitoring information is updated, and can be viewed using *tcov*. This approach differs from that of *gprof* and *prof* in that only counts and not durations are measured [21].

2.1.6. Adding print statements

The most basic method of monitoring software is the insertion of print statements into the code. With analyzing memory dumps, it is among the earliest methods developed. Since it is so basic, it can provide

the basis for more complex systems, but these must be built *ad hoc* for each target. The limit on information about system calls is also not covered. The output is also slow, unless special efforts are made to affect the buffering.

2.1.7. Limitations of Existing Unix Tools for Software Monitoring

While Unix provides a reasonable assortment of monitoring tools, the tools themselves are not designed to work together and there are major gaps in its coverage. No information about system usage by individual processes is provided, nor is any method of relating an individual process to the system as a whole.

2.2. Previous Work by Other Researchers

The use of the data produced by monitoring as a database has been suggested by Garcia-Molina *et al.* [5] in their proposal for debugging a distributed computing system. They propose gathering data through the normal software monitoring approach of installing sensors. Their approach differs from traditional monitoring approaches by treating the data produced as a single relation in a database. This has the tremendous advantage of using existing database software for developing analytic tools. Their prime interest is in providing and examining traces of transactions in a distributed system.

Goldberg and Popek use software monitoring in their analysis of a distributed file system [7]. Their purpose was to use the data gathered for evaluating the relative merits of a distributed file system versus a local file system, and for optimizing the distributed system. Their interests lie in performance measurements, so that their sensors are used to determine the execution time of functions. The system they use is not intended to be a general monitoring system.

Miller, Macrander, and Sechrest developed a monitoring system for metering distributed programs [11]. Their approach was to instrument the communications and process control routines in the Unix kernel with metering controlled by daemons. Processes to be metered are specified to the daemon, which creates a filter process to handle the metering data. Programs to analyze the data must be written by the user. User programs to be metered by the system require no modification, nor can additional metering be added to a program. While their method is extensible to other events in the kernel, it is not intended to meter process events that do not occur in the kernel and is not readily extensible to handle such metering. This

system is clearly meant to provide a means of debugging and studying distributed programs on a case by case basis, rather than to analyze the behavior of the entire sytem.

Kupfer discusses a method of remote procedures call for implementing monitoring tools and uses *vmstat* as an example. This is a study in remote monitoring implementation using an existing protocol rather than an actual monitoring system. [10]

Concurrent with the development of the project presented in this thesis are projects by Ousterhout and Floyd. Ousterhout *et al.* instrumented the Unix file system and created tools to analyze the data produced [12]. Specific sensors were created to monitor events in the file system for answering specific questions about the file system. The thrust of their investigation was to use monitoring to investigate specific behavior of the Unix file system, rather than to illustrate a general monitoring approach through such an investigation.

The approach by Floyd [4] also entailed instrumenting the file system. The thrust of his investigation is the implications of the use of files and directories for designing and evaluating distributed file system implementations. To accomplish this a tool was built for characterizing the usage of files by the examination of data produced by the sensors, allowing the selection of data based on file usage. Statistical tools were used to examine the resulting breakdowns, so that relative usage of file types could be compared. A set of library routines associates related sensor data, such as that for open and close operations, to relieve the analysis programs of that burden. It differs from the approach taken here in not being a generalized set of tools, but rather a specific set designed to answer specific questions.

2.3. Overview of Thesis Project

The approach to software monitoring described in this document is to provide a general purpose framework on which experiments can be devised, using an instrumented Unix file system as an illustration of the method. The system consists of a data collection component and a data analysis component; the diagram in Figure 1 presents the data collection part of the system. The kernel was modified to contain *sensors* that detect events in the kernel and store *event records* in an event record vector, and a *Monitor* system call to manage the vector and sensors. The Monitor call is the agent for controling the enabling of sensors in the kernel, transferring event records from the vector to the *Accountant*, adding event records

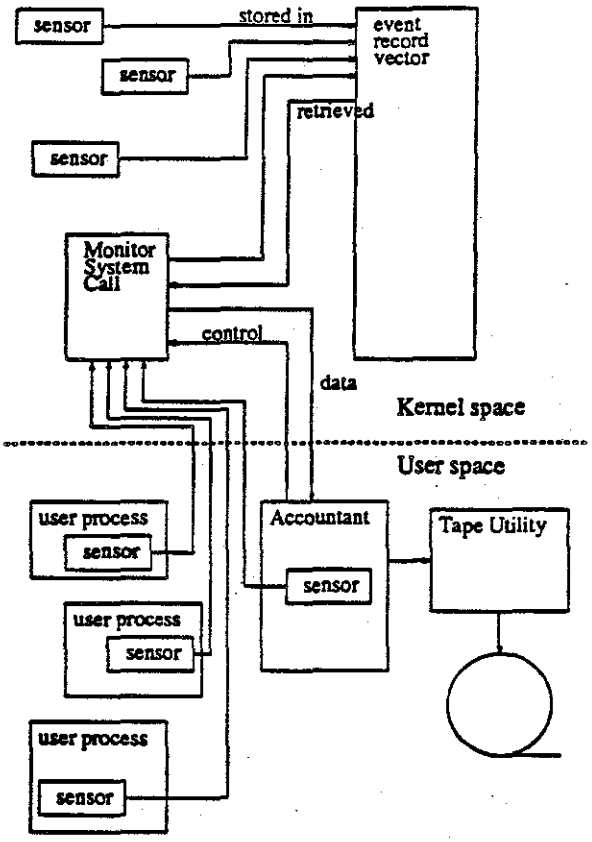

Figure 1.

from user processes to the event record vector, and handling communication between the Accountant and user processes. The Accountant user process controls accounting through the Monitor, and retrieves the event records stored in the event record vector from the Monitor for storage in disk files, where a tape utility transfers them asynchronously to tape. The Accountant and other user processes can also contain sensors that use the Monitor to record their event records in the event record vector.

The analysis process is illustrated in Figure 2, taking the tapes prepared in the data collection phase and passing their data through a pipeline. The program *enschema* associates the event records with a schema describing those records that was derived from the sensor descriptions to a data type called a *stream*. The *tool pipeline* is a series of relational operations performed on the stream and is assembled from the tools discussed in Chapter 7 and defined in Appendix B.

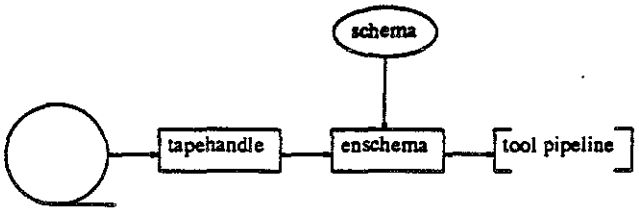

Figure 2.

The monitoring system is part of the SoftLab project at the University of North Carolina, Chapel Hill. Other related parts of SoftLab are a compiler for the sensors and a temporal query language for relational databases [17].

The system is designed to run under Unix 4.2 BSD. It has been run on Vax minicomputers and on Sun-2 workstations at UNC, Ohio State University, and at Carnegie-Mellon University. At C-MU, modifications are being made to run the data gathering portion of the system in a multiprogramming environment using graphical tools to examine the data. This will be discussed in more detail in the appropriate chapters.

Chapter 3

Event Record Data Structures

This chapter covers the data structures used for the event records, detailing the design considerations and the implementation.

3.1. Constraints

The Unix kernel on Sun workstations cannot exceed 512 Kbytes. This places an absolute limit on the space available for storing event records and requires that event records must be efficient in using space. Hence variable length format for the event records is used.

An event record consists of an initial structure that contains an identifier for the type of record and the length of the record in short integers. The remainder of the event record is an array of the C type `short` that is cast to the appropriate type. The decision to use type `short` rather than type `char` was made based on the typical size of operands and the efficiency of assignments. The sensor specific fields in event records will be covered in the next chapter along with the sensors that generate them.

3.2. Event Buffer

The kernel sensors deposit event record in an event record vector organized as a circular queue with pointers to the beginning and end, a count of how much of the vector contains, and a flag to indicate if the vector is full. The event vector is physically composed of `unsigned char` (the Unix kernel uses the `typedef` facility to call this `u_char`), but is logically treated as consisting of `short` integers. This allows arbitrary handling of data within the event vector without concern for object alignment while conforming to the event record format. The Monitor copies the event record vector to the Accountant on request as an array of integral event records, and then resets the circular queue pointers, counts, and flags.

3.3. Streams, Tuples, and Schemas

The data produced by the sensors is gathered by the Accountant and written to a series of files as raw event records. The Accountant stores the event records in binary to conserve disk space. A *schema*, produced when the sensor descriptions are compiled, is required to interpret the event records. The schema describes the event records based on the defining sensor and provides the means for identifying and labeling event records and their fields.

The combination of a schema followed by the event records it explains is called a *stream*. A stream can be considered a relational database, with each event record in it a tuple with the relation determined by which sensor created it. As such, the fields within the record are domains in the relation. Certain domains are necessary to identify the relation and are immutable. These are the domains *cmdtype*, *cmdlength*, and *eventnumber*. The last is also known as the *sensor id*.

3.4. Implementation

An event record consists of a head structure, mon_cmd, describing the type of sensor that generated the record and the length of the record, and a body, containing fields specific to the sensor type and definition.

```
struct mon_cmd {
        char type,
             length;
        };
struct mon_pevt {
        struct mon_cmd cmd;
        short          eventnumber,
                       performer;
        long           object;
        short          initiator;
        long           timestamp;
        short          fields[EVENT_LIMIT];
        };
typedef struct mon_pevt mon_putevent;
```

The event record is considered to be an array of **short**. The structure mon_cmd is used rather than just having to separate fields so that assignments can be more readily made. Other types of events share mon_cmd but fill the rest of the event record with their own fields. Since all records share mon_cmd, they can be read and written without regards to their identity, based solely on the length. The appendix *Generating Standard Sensors* details how to handle the variable length and **short** alignment constraints

for character strings.

Since the records are variable length, some additional complexities arise when handling them in the event vector. In implementing the event record as a circular queue, a static array was used to save the overhead of memory and linked-list management. This means that before we can place a record on the queue, we must see if and where wrap-around takes place in the record. Instead of checking for every short integer in the record, an extra space, the queue has an *appendix* at its end large enough to hold the longest event record. This allows a single test to take place after the record has been copied into the vector. If wrap-around is necessary, a routine copies the part of the record in the overflow area to the front of the event record vector and adjusts the write pointer. The wrap-around routine only executes once per pass through the queue, so the overhead is small.

```
u_char *mon_write_ptr,              /* Write pointer in mon_eventvector   */
       *mon_read_ptr,               /* Read pointer in mon_eventvector    */
       *mon_eventvector_end;        /* First pos after buffer, start of appx */
int    mon_eventvector_count;       /* No. of chars of valid event records
                                       in mon_eventvector                 */
int    mon_oflow_count = 0;         /* Event record overflow              */
u_char mon_eventvector[MON_EVENTVECSIZE + MON_EVENTRECSIZE];
                                    /* Event record ring buffer           */
```

A further problem is encountered when copying event records from the queue to the accountant. The accountant requests the data in terms of an absolute size, and receives as many event records as will integrally fill this size. If the queue contains more data than can returned to the accountant, the monitor routine must advance through the event records until it finds the appropriate number to return. In addition, wrap-around must also be handled so that the accountant receives the event records in the proper order. The code to handle this will be discussed in Chapter 5.

A schema is organized as a hierarchical structure, called a `database`, since it permits treating event records as tuples in a database. The format of all schemas is shown in the following IDL code, and can be found in the file *schema_idl.idl*.

```
Structure schema Root database Is
     database  =>  database_name  :   String,
                   relations      :   Seq Of relation;
     relation  =>  rel_name       :   String,
                   rel_sensor_id  :   Integer,
                   rel_vlensensor :   Boolean,
                                      -- true if variable length
                   attributes     :   Seq Of attribute;
```

```
        attribute =>  attr_name       :   String,
                      attr_length     :   Integer,
                                          -- in bytes
                      attr_pos        :   Integer,
                                          -- from beginning,
                                          -- < 0 if notfixed
                      attr_type       :   type;
        type      ::= type_integer | type_rational |
                      type_string  | type_boolean;
        For type Use Enumerated;
End
Process schema_idl Inv schema Is
        Pre  input  : schema;
        Post output : schema;
End
```

The format compiles into a set of structures, macros, and routines that the analysis tools (Chapter 6) use to handle the event records. Every sensor compiled by the compiler has a `relation` structure generated corresponding to the sensor's event record. The schema allows event records to be processed by name without building into each tool knowledge of all event records and sensors. If a tool changes the format of an event record, it must also adjust the schema to reflect the change.

A schema is a `database` structure, containing a database name and a linked list of `relation` structures. This list contains a `relation` structure for all possible relations that an accounting session can generate, though programs processing the event records can generate additional relations.

Each `relation` structure contains a name, `rel_name`, taken from the name of the sensor, an identification field, `rel_sensorid`, taken from the `eventnumber` field, `rel_vlen_sensor`, a flag indicating if the associated event record is variable length, and `attributes`, a linked list of the domains in the relation. A new `relation` can be created from an old by changing its name and identification. A `relation` can also be changed back and forth from variable to fixed length. The contents of the `relation` can be altered by changing the `attribute` list.

A domain is an attribute of a relation and an `attribute` structure represents the domain in a `relation` structure. The `attribute` contains a name and type, `attr_name` and `attr_type`, defined in the sensor definition, its position in the event record, `attr_pos`, and the length of the domain field, `attr_length`, that varies with the type of the field. The offset of a domain within an event record may vary depending on whether the associated field occurs after a variable length field. If this is the case, the offset is recorded as a negative number and must be reset each time an event record in that relation is read.

The length is fixed for all types of fields except for character strings, where the length is variable, but always a multiple of two, so that the organization into an array of `short` is not violated. A field with a length of only one byte must be paired with a similar field or a padding byte also to ensure this organization.

Since the schema must preceed the event records, any changes to the schema must be known before the first event record is written out, since the schema must be written first. To support the stream concept, a data structure `Mstream` (for *M*onitor *stream*) exists, which associates a Unix `FILE` structure with a schema.

```
typedef struct S_Mstream {
            FILE     *fp;
            char     record[MAXRECSIZE];
            database schema;
            short    flag;
            } Mstream;
```

There is a one to one correspondence between `Mstream`*s* and `FILE`*s*. The `Mstream` element `fp` is a pointer to the associated file. Each `Mstream` structure contains room for the incoming event record in its element `record`. Note that, unlike event records, this is an array of type `char`. There is also an element for the stream's schema, and a flag to indicate if the schema has been read or written. This allows the schema to be automatically read or written on the first appropriate operation, and prevents duplicate writes.

```
typedef struct s_tuple {
            char     *record;
            relation relation;
            } tuple;
```

A `tuple` is defined as an association of an event record and its `relation`. When it is read from a stream, `record` points to the `Mstream's` record and `relation` is the `relation`in `Mstream's` schema for that event record. The `relation` is ignored when the `tuple` is written.

Chapter 4

Sensors

This chapter covers the sensors used to instrument the kernel, concentrating on the design of the sensors, the choice of sensors, and their placement in the kernel.

4.1. Basic Sensor Design

For any event record, certain basic information is needed. We must know the type of sensor that created the event record – such as a kernel sensor, a user sensor, or an error record, to give three examples. Since we are using variable length records to save space, we must specify the length of the event record. To know what fields are in the record, we need to have an event number that is specific to each sensor. For kernel sensors we want to know who performed the action and who initiated it by saving the process identifier (*pid*) of those two processes. Most sensors will have some object that they are recording information about. A unique identifier for this object is stored. Much analysis requires knowing the time that an event happened, so a timestamp field is included. This yields the following standard fields:

type of event record	1 byte	what created the record
length of event record	1 byte	in short integers
event number	2 bytes	defined by the sensor compiler
performer	2 bytes	who performed the sensor
object	4 bytes	what was being affected
initiator	2 bytes	who requested the action
timestamp	4 bytes	when did this occur

In addition, free form fields follow these in the event record that can be used for sensor specific data.

Sensors are defined via a Sensor Definition Language [16] and compiled either by a special compiler or by hand, as defined in *Generating Standard Sensors* (see Appendix A). The sensor compiler produces sensor include files containing macros and declarations for the target programs and a schema for the event records that the sensors create. The schema is used by the analysis tools to interpret the event records.

4.2. What Do We Want to Find Out?

The choice of what specific fields to put into each sensor is determined by what questions we may want to ask. We have identified the following pieces of data that would be useful to collect.

within each process
- how many files are open over time, and at one time
- impact of process on system: percentage of total calls

volatility
- reads to writes ratios
- modes of read, write, read/write
- relative sizes of reads and writes
- creates, deletes

event frequency
- proportion of total events for each event type

locality
- length of paths to files

within each file
- frequency of operations
- size of reads, writes
- duration of access
- number of reads & writes per access
- file size

general
- relationship of any of the above to the load average, number of users, etc.

4.3. Overview of the Unix File System

The Unix file system consists of a tree of special files called *directories* that can contain both regular files and other directories. A process has associated with it a *current working directory*, whose files can be specified by name. Files not in the current working directory are specified by a *path*, that is absolute from the root of the tree or relative to the current working directory. The most common primative operations on a file are `open`, `close`, `read`, `write`, `seek`, and `creat` (*sic*). `open` follows a path and returns a *file descriptor*, that is an entry in a system table. The path is handled one component at a time and is a measure of the locality of reference. `close` removes the reference to the system table. Files are deleted when there are no references in the system table and when no directories point to it. `read` and `write` always do implied `seeks` before operating, so `seek` can be traced by recording the position in a file where `read` and `write` take place. Files can exist in more than one directory, but can be absolutely identified by the more basic Unix I/O system. A file is actually a specific *inode* on a specific *device*. An inode is a data structure that records the physical location of the file. Each device contains its own inode list numbered from zero to a limit based on the capacity of the device. [19]

4.4. The Sensors

The following sensors, presented in Sensor Definition Language, enable us to determine the information that was listed above. The object for these sensors, when it exists, is the device/inode pair, that uniquely identifies the file.

FileClose

```
Event FileClose (Device, INumber:doubleinteger object;
                 FinalSize:doubleinteger
                 ) is
      timestamped, sensortraced;
```

The FileClose sensor is triggered in the kernel routine closef in the file *kern_descrip.c* whenever a file is closed either by a process or by the system. Its event record identifies the file and the size of the file when closed.

ReadSensor

```
Event ReadSensor (Device, INumber:doubleinteger object;
                  FilePos:doubleinteger;
                  ActualCount:integer
                  ) is
      timestamped, sensortraced;
```

ReadSensor records reading from a file. Its object is the file in question. FilePos is the position in the file and ActualCount is the count of bytes read. This sensor provides basic information on activity on open files, along with WriteSensor. It is installed in the routine rwuio in the file *sys_generic.c* and is invoked by the low level read system call.

WriteSensor

```
Event WriteSensor (Device, INumber:doubleinteger object;
                   FilePos:doubleinteger;
                   ActualCount:integer
                   ) is
      timestamped, sensortraced;
```

WriteSensor records writing to a file. The arguments to WriteSensor are the same as to ReadSensor. It is also installed in the routine rwuio in the file *sys_generic.c* and is invoked by the low level write system call.

OpenSuccessful

```
Event OpenSuccessful (Mode:integer;
                      InitSize:doubleinteger
                      ) is
      timestamped, sensortraced;
```

OpenSuccessful records the opening of a file. It does not name an object, the name is provided by the previous string of event records produced by NextComponent. The Mode parameter records the access mode in which the file was opened and the InitSize parameter records the size of the file when it was opened.

NameStart

```
Event NameStart (Device, INumber:doubleinteger object) is
      timestamped, sensortraced;
```

`NameStart` is used to signal the beginning of the lookup of a filename. It records the file identity for the beginning of a pathname in its object. It is installed near the entry of the pathname lookup routine `namei` in the file *ufs_nami.c*.

`NextComponent`

```
Event NextComponent (Device, INumber:doubleinteger object;
                     FileName:string[127]
                     ) is
      timestamped, sensortraced;
```

`NextComponent` records in its object the file identity of the current pathname component, which it stores in `FileName`. `NextComponent` is installed in the pathname lookup loop of the routine `namei` in the file *ufs_nami.c*. The number of `NextComponent` event records for a process is a measure of the locality of reference for the file. The object of the `NextComponent` event record immediately preceding an `OpenSuccessful` event record for the same process is the opened file returned to the process.

`INodeCreate`

```
Event INodeCreate (Device, INumber:doubleinteger object) is
      timestamped, sensortraced;
```

`INodeCreate` signals that a new file has been created. The object is that file. `INodeCreate` is placed in the inode allocation routine `ialloc` and can be triggered by an `open` or a `creat` system call.

`INodeDelete`

```
Event INodeDelete (Device, INumber:doubleinteger object) is
      timestamped, sensortraced;
```

`INodeDelete` shows that its object, a file, has been deleted. It is placed in the inode recovery routine `ifree`. This is part of the housekeeping done by the system, and is not directly called by user processes. Files are only deleted when there are no more references to them.

4.5. Testing

Initial testing of the sensors was done by emulating the affected routines in the kernel with a set of routines that filled in constant values into the structures that the sensors referenced. These emulated routines were compiled with the Monitor (Chapter 6) and a truncated version of the Accountant (chapter 5). This permitted full testing of the sensor control code and verification that the values were put into the correct

places in the event records while not affecting an operational system. The kernel did not have to be modified for each code change nor was rebooting necessary after aborted tests, greatly speeding the testing process.

It was more difficult to test the validity of the data gathered from the kernel itself. When the monitoring system was enabled on a system with a normal load, data was generated at the rate of up to 50,000 bytes of data in three seconds. This volume made it difficult to determine what was really happening. Verification of the sensor data consisted of two parts. The first determined what was happening in the kernel to insure that the sensors were placed properly and thus monitoring the desired events. This was done by adding `printf` statements to the kernel to trace the execution pattern and tracking this pattern against that produced by the sensors. Only small test runs could be performed with this method since it produced disastrous effects on kernel performance and reliability. The second part of verification insured that the fields accessed by the sensors contained the desired information by using a test suite of processes. The suite accessed specific files in a specific order so that the event records generated and their values could be predicted. Both parts of the verification required a primitive print program, `blindprint`, to view the event records produced by the test runs.

Additional verification has come by having the system run at other sites, namely the Computer Science departments at Ohio State University and Carnegie-Mellon University. The work at those sites aided greatly in debugging the code.

4.6. Future Work

4.6.1. IBID sensor

A standard sensor takes up a certain minimum amount of room, approximately 20 bytes. When instrumenting routines that are accessed frequently, this overhead can become excessive, especially since the data in these circumstances is often only slightly different from event record to event record. The overhead can be minimized by having a sensor generate event records that take advantage of consecutive calls. These event records can either be completely new types of records or variations on existing types. Two new types of records are proposed: *IBID* records, that record only fields that differ from the preceding event record, and *count* records, that have a count field that is incremented for consecutive invocations of

the same sensor. Instead of using new types of event records, normal event records can be altered to accomplish the same tasks. For either new event records or old, criteria must be established for using the abbreviated form rather than the full form of an event record. Only those fields that differ from the previous event record need be stored in the new structures, but to reduce complexity, the space saving structures will only be used if the previous event record was generated by the same sensor and if they share the same object, performer, and initiator. This would happen frequently with common operations like `read`. The fields that need to be stored may be indicated by `ibid` in the sensor definition, while the remaining field, those not associated with `ibid`, are only recorded when the previous event record did not meet the IBID criteria. The `timestamp` field, when active, is always generated. An example is

```
Event ReadSensor (Device, INumber:doubleinteger object;
                  FilePos:doubleinteger ibid;
                  ActualCount:integer ibid
                  ) is
      timestamped, sensortraced;
```

Extra control information is added to the sensor to detect whether to save space or to use the regular event record. The `object`, `performer`, and `initiator` fields are at fixed points in the previous event record and can be compared directly by keeping one additional pointer to the previous non-space saving event record, which would be cleared when the event vector is cleared, and reset when the regular sensor code is used. A bitmask determined at sensor compile time is used to specify those fields that are to be updated by the space saving code. The fields that are not flagged by the bit mask are copied from the previous event record at analysis time. While this could be done for all sensors, it adds overhead for sensors unlikely to be called successively.

```
struct ibid_event {
        mon_cmd cmd;            /* type = MONOP_PUTEVENT_IBID   */
                                /* length = variable            */
        short   new_fields;     /* which fields to replace      */
        short   fields[254];    /* values                       */
        };
```

If the sensor is not timestamped, then a *counter* event record can be used, that counts the number of identical event records that occur. A sensor generating a counter event record requires the same initial logic to determine what type of event record to generate. If a counter record can be used, the sensor requires less

time to generate it than either an IBID record or a regular record. Since the counter record only contains one data field, it requires much less space than either alternative.

```
struct count_event {
        mon_cmd cmd;            /* type = MONOP_PUTEVENT_COUNT */
                                /* length = 3                  */
        int     count;          /* times to replicate event    */
        };
```

The advantage of using separate count and IBID event records is that they do not conflict with data already present. They can be merged into the same format as regular event records by a simple tool at analysis time. The disadvantage is that there is still a significant amount of overhead even in these sensors. There is also more complexity when handling the interface with other sensors, since there can be many IBID records between the last regular record and the current record.

An alternative approach to exploiting consecutive calls to a sensor is to add structures to the event record. These structures take the form of a overhead `struct` containing a count and the bit mask of target fields, and an array of the target fields. This approach removes all the overhead of extra event records, by simply expanding the existing ones. Only the last event record written needs to be examined to determine the sensor's action. Each update expands that record, changes its length, and moves the event vector's write pointer.

```
struct ReadSensor {
        mon_cmd  cmd;           /* type = MONOP_PUTEVENT_IBID  */
                                /* length = variable           */
        short    eventnumber;
        long     object;
        short    initiator;
        short    performer;
        long     timestamp;
        long     filepos;
        short    actualcount;
        struct ibid {
                 u_char ibid_count,
                        ibid_mask;
                 } ibid_head;
        short    fields[];      /* sets of timestamp,filepos,actualcount */
        };
```

A disadvantage of this approach is that the format is only loosely compatible with the older format. It can be converted to the older format, though, without much difficulty, since all the data is present. It has the

great advantage of having very little overhead, and of having but one format for both IBID and counting records. The latter is handled by having a null `ibid_mask`.

4.6.2. Use of Signals to Coordinate with the Accountant

Using timing loops to retrieve event records from the system has the problem that either many calls will be unnecessary or the buffer will overflow. This situation can be corrected by signaling the accountant when the buffer is ready to be read. The signaling is done by the kernel sensors themselves and by the monitor routine for user sensors. The sensor definition is changed to detect when the buffer is nearly full and to then signal the accountant. The sensors would still check for overflow. This requires a flag to indicate that the accountant has been signaled and a threshold to specify when the signaling should occur. When signaled, the accountant must interrupt what it is doing and retrieve the records from the monitor.

4.6.3. Better Concurrency Checking for Multiprocessing

The monitor system call is a critical section [1]: the event record vector must only be accessed by one process at a time, otherwise the pointers will become corrupted. This is not significant on a uniprocessor running regular Unix, since only one process is active at a time and interrupts can be disabled to insure completion of sensors and the monitor routine. On a multiprocessor these measures do not suffice. A functioning semaphore must be used in both kernel sensors and the monitor call to insure exclusivity.

4.6.4. Network File System

Sun's *Network File System* (NFS) [25] presents other problems. The information desired for a sensor may not be kept in the local system, and must be requested remotely. Since the information is not locally present, additional changes must be made to the kernel to provide the information. If this is done outside the control of a sensor, the new code could be executed at an inappropriate time, such as when rebooting, and cause system failure. If the code for providing the information is put in as a parameter to the sensor macro, the remote request generates an additional `read` system call, that in turn triggers `ReadSensor`, threatening to become infinitely recursive. A possible solution is to define a new set of sensors for NFS that are placed in the NFS code itself, rather than the interface to the standard file system implementation.

Chapter 5

System Call - the Monitor

The *Monitor* is the local system call `SYSL_MONITOR` that is used by the Accountant to manage the event record vector and the kernel sensors, and to communicate with user processes.

All data from the sensors is written into the kernel's event record vector. To protect the operating system, data in the kernel is only accessible to user processes through system calls. Since the Accountant, a user process, controls the sensors in the kernel and reads the event record vector, a system call must be provided as an agent.

To ease adding system calls to the kernel, an indirect system call, `SYSLOCAL`, with its own table of system calls, was added to the table of system calls. To invoke the Monitor requires calling `syscall` with the entry number in the system call table for `SYSLOCAL`, the entry number in the local call table for the Monitor, and the address of the buffer holding the command to the Monitor. The include file *syslocal.h* contains the entry numbers for `SYSLOCAL` and `SYSL_MONITOR`.

```
syscall(SYSLOCAL, SYSL_MONITOR, (unsigned char *)&preq);
```

5.1. Function

The Monitor maintains a vector of enable bits, `mon_enablevector`, which controls the status of the kernel sensors, and a message buffer, `mon_requests`, for communication between the Accountant, other user processes, and the Monitor itself. The enable bits in `mon_enablevector` for a specific sensor are determined at sensor compile time (see the appendix *Generating Standard Sensors*). The Monitor enables and disables sensors by switching the appropriate bits as directed by the Accountant through a `MONOP_PUTREQ` command. The communication system is only partly implemented. It handles communication between the Accountant and Monitor but not between the Monitor and other user processes.

The Monitor also manages the event records produced by all sensors. On detecting that the event buffer is full, The Monitor inserts an error record into the buffer to signal that data may be lost. This is only done before writing the event buffer to the Accountant. Event records are only written to the Accountant on request.

5.2. Decoding a Command

The operation of the Monitor is performed in a critical section. This prevents conflicts when manipulating the pointers in the event record vector. The first step in decoding a command is to enter the critical section. In the uniprocessor implementation this is simply a test on a counter, `mon_semaphore`. This counter must be zero for the operation to take place. If for some reason it is not zero, then a sensor was interrupted. This should never happen. It means that the sensors are placed incorrectly or that the kernel has been corrupted. The only action taken is to disable all sensors and to return the value MON_CONCURRENCY_ERR. This kernel should then be fixed, recompiled, and rebooted. In any case no more accounting can be done. As its name implies, `mon_semaphore` may be replaced by a true semaphore in a multiprocessing implementation. It would then wait until it could enter the section before proceeding.

A system call must copy its parameters from user space to kernel space to access them, and must copy its return values from kernel space back to user space. Since the commands to the Monitor are variable length, the Monitor must first determine the size of the command. All commands to the Monitor begin with the C `struct` mon_cmd (see Chapter 3), which contains the length of the entire command. This structure is copied first and its length component is used to determine the size of the command. The command is then copied in its entirety for the length specified in mon_cmd. This does mean that mon_cmd is copied twice, but it is only the length of a `short` and reading the command in its entirety enables any padding in the command structure to be ignored and obviates including machine and compiler dependent code to determine padding within structures. A `switch` statement uses the `type` component of mon_cmd as the argument to determine which command to process. After the processing the command, the call exits the critical section and returns.

5.3. Command Operation

The function of the different Monitor commands are found in the manual pages for in Appendix B. The internal workings of the commands are described here. The commands themselves are represented as integers and are defined in the file `monops.h`.

Most commands are executable only by the Accountant. Exceptions are `MONOP_GETREQ`, which a user process issues to retrieve messages from the Accountant, and `MONOP_INIT`, which can be executed only when there is no Accountant. In an emergency, the super user may also issue `MONOP_GETEVENTS` and `MONOP_SHUTDOWN`. This feature is provided to prevent the loss of data should the Accountant process prematurely end. If a process tries to issue a privileged command, the Monitor returns the value `MON_NOT_ACCTNT`, found in the file `monerrcds.h`.

If the Monitor receives a command other than `MONOP_INIT` before accounting has started, it returns the value `MON_NOT_INIT`. If `MONOP_INIT` is issued after accounting has started, then the Monitor returns `MON_ALRDY_INIT`.

`MONOP_INIT`

This command initializes accounting and the process that issues the command becomes the Accountant. The supporting variables for the event record vector are initialized:

`mon_write_ptr`	write pointer into the event record vector
`mon_read_ptr`	read pointer into the vector
`mon_eventvector_end`	pointer to the end of the vector
`mon_eventvector_count`	byte count of the filled portion of the vector
`mon_oflow_count`	count of attempted writes after the vector filled

Since the current state of the sensors is not known, the vector containing the sensor enable bits, `mon_enablevector`, is set to all zeros. The buffer to communicate with other user processes, `mon_requests`, is cleared of old requests. The size of the event record vector in bytes, which is a compile time constant, is returned to the calling process.

`MONOP_PUTEVENT_INT` and `MONOP_PUTEVENT_EXT`

These commands write event records into the event record vector. The process id of the caller and the time when the command is processed are written into the event record in the command. The writing of the event record is performed as a sensor would write it (see Chapter 4 and the Appendix, *Generating Standard Sensors*). If writing the record would overflow the vector, `mon_oflow_count` is incremented and the Monitor returns `MON_BUFF_FULL` to the calling process.

`MONOP_GETEVENTS`

This is the most complex of the commands. It handles transferring the event records from the event record vector to the Accountant. The complexity arises from having to adjust the amount of data transferred to fit the amount requested by the Accountant while handling vector wrap around and

insuring that only integral records are transferred. The first task of MONOP_GETEVENTS is to determine if any data was lost because of a full vector. If this occurred, an error record is written into the vector.

```
struct mon_erec {
        struct mon_cmd cmd;
        long           val;
        };
typedef struct mon_erec mon_errrec;
```

The field val receives the count of records missed because of overflow. There is always room left in the vector for a single error record. Two cases arise when determining how much data to transfer. If more data is in the vector than requested, then the lengths of the records in the vector are accumulated in the *character count*, the total amount to transfer to the Accountant, by moving from mon_cmd to mon_cmd until the requested amount is satisfied. If this process reaches the end of the event record vector (wrap around) before satisfying the request, the Monitor handles it by continuing the process from the beginning of the vector. If the accountant requests more data than is available, the entire vector can be transferred without handling event record boundaries. The size of data in the event record vector becomes the character count. The data must be transferred in the order that it filled the vector. Wrap around divides the data into two parts and forces the transfer to be done with two calls to copy. Wrap around is detected when the write pointer has a lower address than the read pointer. If there is wrap around, the *transfer count*, the amount to transfer in a single call to copy, is set to the minimum of the character count and the length between the read pointer and the end of the vector. This amount is transferred to the Accountant's buffer, the character count is decremented, and the pointer into the Accountant's buffer is incremented. If the character count is not zero, then data from the lower addresses for the lower amount is transferred. All the pointers, counts, and flags are updated, and the Monitor returns the amount of data transferred.

MONOP_PUTREQ

The Accountant controls sensors in both the kernel and user processes with MONOP_PUTREQ commands.

```
struct mon_request {
        short targetpid,
              eventnumber,
              enablevalue;
        };
struct mon_preq {
        struct mon_cmd     cmd;
        struct mon_request req;
        };
typedef struct mon_preq mon_putreq;
typedef struct mon_preq mon_getreq;
```

Kernel sensors are specified by a targetpid of zero. If the kernel is the target, the bit specified by eventnumber in mon_enablevector is set to the value of enablevalue. Requests to user processes are not handled directly, but are stored in a static sized array of mon_request structures, mon_requests. The request is stored in the first available slot in mon_requests,

and the user process is signaled that a request is awaiting it.

MONOP_GETREQ

When signaled that a request is awaiting it, the user process issues a MON_GETREQ to the Monitor to read the request. The Monitor searches the request vector for the first message with a matching process id. No allowance is made for multiple messages. Since multiple signals are lost in Unix (see *signal*(2)), there is no easy way to inform a process that there are multiple messages, though a process could continue to ask for requests until no more are returned. When the message is found, the message is returned to the caller and the slot in the request vector is cleared. If no message is found, then an error record is added to the event record vector and the Monitor returns MON_REQ_NOT_FND to the caller.

MONOP_SHUTDOWN

The Accountant issues this command when accounting is finished. All sensors should be disabled and a final MONOP_GETEVENTS issued before shutdown so that no data is lost. Shutdown disables any kernel sensors that might have been left enabled and resets all pointers, flags and counts.

5.4. Testing

Initial testing of the Monitor was done using stub routines for certain system calls and an abbreviated form of the Accountant (see *acct* (1L) in the Appendix). This allowed testing to be done in user mode using interactive debugging. The abbreviated form of the Accountant was used to minimize any side effects from errors in the Accountant. This was not sufficient for testing since it did not provide for asynchronous calls, or for problems with the kernel interface.

Initial testing in the kernel was done with a reduced accounting environment. The minimal Accountant was again used, only a single sensor was placed, and a reduced event record vector was used to isolate errors. Since errors in the Monitor would cause the system to crash, the debugging was carried out on workstations. This minimized interference with other users and permitted more rapid compiling and rebooting. Since dynamic debugging cannot be used on the kernel, execution was traced with print statements. The test system was put under load by a file system exerciser designed by M. Satyanarayanan [14]. The event vector logic was put under load by using parallel unbuffered I/O. The code for communicating with user processes was not tested.

A beta release of the system was installed at Ohio State University and at Carnegie-Mellon University. This provided the opportunity to detect more errors in implementation and installation, which have been corrected.

Chapter 6

Accountant

The user process that controls the kernel sensors is called the *Accountant*. The Accountant controls the enable status of the kernel sensors, periodically dumps the kernel's event record buffer to disk, and, orthogonal to this project, communicates with other user processes that have sensors. This chapter will describe the Accountant's function for this project, the internal operation to accomplish these functions, and a typical session of using the Accountant. This will be followed by a brief delineation of the use of the Accountant with other processes with sensors, and by a section discussing future plans for the Accountant.

6.1. Standalone Function

No *accounting*, which is the recording of event records, takes place except under the supervision of the Accountant. The Accountant must first initialize the accounting session, initialize all desired kernel sensors, and, when the session is finished, disable all sensors. These are all accomplished through the mediation of the Monitor (see Chapter 5, above). In the event that the Accountant dies, the sensors can be disabled manually by the utility *shutdownacct*(8L) (see Appendix B) but the data left in the event record buffer are lost. If the sensors are left running, they will eventually run out of room in the event record buffer. The kernel sensors will then cease to record data, and user sensors will get an error from the system call. A new session cannot be started until accounting is shut down.

The Accountant's second task is to periodically read the event record buffer via the monitor system call. The size of the buffer is returned to the Accountant when it initializes accounting. It receives integral event records from the monitor that are written out to the current output file.

Since output may be extremely large, the data will have to be spooled to tape during the longer accounting sessions. To facilitate this, the Accountant periodically changes output files, so that the old ones can be

accessed by a tape utility (*dd*(1) will suffice for the tape utility). The size of file for switching is defined at compile time and is currently based on the Unix 4.2 BSD parameters.

6.2. Internal Operation

All communication with the kernel is done through commands to the Monitor sent by the local system call, `SYSL_MONITOR`. The values of `SYSLOCAL` and `SYSL_MONITOR` are found in the file *syslocal.h*, which must be included in the C file for compilation. For the formats of the various commands, see the manual page for *SYSL_MONITOR*(2L) in Appendix B.

```
syscall(SYSLOCAL,SYSL_MONITOR, (unsigned char *)&command);
```

The procedure `InitAcc` is called by `main` to initialize the accounting session. `InitAcc` calls the procedures `InitOutput`, `TurnOnAllSensors`, and `DoUnixProto` to complete the initialization process. `InitOutput` is used first to create a unique file to hold the event records produced by the accounting session. `InitAcc` readies the Monitor by sending the MONOP_INIT command (see Chapter 5) to it through the SYSL_MONITOR system call. The Monitor returns the number of bytes in the event vector. Immediately after initializing the Monitor, no sensors are active. `InitAcc` sends the command MONOP_PUTEVENT_EXT to the Monitor containing a header record for the accounting session. This header contains information identifying the files containing the operating system and the Accountant, and information about the status of the system, such as the load average, number of users, and the time. An optional string on the Accountant's command line can be included in the header record. Since it is the first record in the event vector, it is at the front of the data produced by the session, and serves to separate accounting sessions in a stream of event records. `TurnOnAllSensors` enables individual kernel sensors as indicated by an array of sensor ids, `ActiveSensors`. For each entry in the array, a MONOP_PUTREQ command with an enable value of one is sent to the Monitor. Finally, `DoUnixProto` is called to read any accumulating records in the event vector and to write them out to disk. The event records are gathered through the MONOP_GETEVENTS command sent to the Monitor, and are written out to disk by the procedure `WriteEventRecord`. The MONOP_GETEVENTS command specifies a buffer address and a the buffers size. It writes into the buffer as many whole event records as the Monitor has available up to the size of the buffer. `WriteEventRecord` is called to

write the records to the file. It manages the amount of data written to the current file and changes files with the procedure `SwitchFiles` after reaching a predetermined size. This file change is hidden from the higher level procedures. `SwitchFiles` closes the current file, generates a new unique file from a template, and sets the current output file to be the newly created file. The old file is now available to be spooled to tape. A set of different files is used instead of a pipe so that the system can be left unwatched. In actual operation on a Sun, the kernel event vector reached saturation in five seconds under heavy load. If the pipe were connected to a tape process that requested action, the Accountant would enter a wait state and the event vector in the kernel would reach saturation long before any operator intervention could take place.

At this point the sensors are beginning to fill the event vector in the kernel. The Accountant returns to the main routine and executes an infinite loop consisting of sleeping for a fixed period of time and executing `DoUnixProto`. The Accountant never ends on its own. Accounting must be stopped by signaling the Accountant with the signal SIG_TERM (see *signal*(3C) in *The Unix Programmer's Manual*). When signaled, the procedure `Finish` is called to end the accountant session. `Finish` invokes `TurnOffSensors` to disable all sensors in the `ActiveSensors` array and processes any remaining records in the event record buffer through `DoUnixProto`. The accounting session is terminated by sending the command MONOP_SHUTDOWN to the Monitor. The current output file is then closed, and the Accountant exits.

6.3. Use

A typical session of the accountant is started by typing the following to the shell (this assumes *csh*(1) is the shell), whose output is in bold. The percent sign (`%`) is the shell's prompt to the user.

```
% accountant "Accounting session example" &
[1] 18105
```

The string is an optional header for the event record stream. The line printed out tells the job number and process id of the Accountant. The Accountant is run for some desired length of time or until some desired event happens. It will produce files in the current working directory that can be spooled out to tape. When it is decided to stop the session, this is typed to the shell:

```
% kill -TERM %1
%
[1] Done          accountant "Accounting session example"
%
```

The files produced by the Accountant can be used by the analysis tools once the proper schema is prepended.

6.4. User Communication

When the Accountant is used as a standalone process, the sensors it enables are determined at compile time. If a different suite of sensors is desired, then the code must be changed and recompiled. The Accountant can also run under a user monitor that controls the accountant through signals. This monitor is called the *Simon* (as in *Sim*ple *mon*itor) monitor. Simon is meant to be used interactively with the Accountant and other user programs using the monitoring system. It allows enabling and disabling specific sensors.

6.5. Future Changes to use Signals

To enable the Accountant to better adjust to the volume in the event record buffer, the sleep–read loop is replaced by a sleep alone that is only interrupted by signal from the Monitor or from the kernel sensors. Upon interrupt, it reads and writes, and then goes back to sleep. Changes to the Accountant entail a new interrupt routine and signal handling, and a slight modification to the main loop.

A difficulty in this approach is testing the cooperative processes. Much of the Accountant could be debugged by simulating the kernel calls in a single user process with the Accountant (see below). Using signals requires two processes rather than one and thus makes testing more complex. A call to a system procedure that was simulated by a call to a user procedure will have to change to some form of remote procedure call. This will make isolating errors more difficult.

6.6. Testing

The Accountant was initially tested by compiling it with a miniature kernel. This mini–kernel contained an emulation of the system calls used by the Accountant, the Monitor system call suite of `syslocal`

and `sysl_monitor`, and a set of skeleton routines to drive the kernel sensors. Since the entire system is in user space, the interactive debugger *dbxtool* (see *dbxtool*(1) of Sun Microsystems' *Programmer's Manual*) was used to trace the Accountant's operation. This tested all the major parts of the Accountant, including initialization, event record gathering and distribution, and shutdown.

Integrated testing of the Accountant using the installed Monitor has been minimal because of changes in the Sun operating system during development. This will be done on a DEC VAX computer running Unix 4.2 BSD.

Chapter 7

Analysis Tools

This chapter discusses the tools used to analyze the accounting data. The Appendix contains Unix manual pages describing how to invoke each tool while the emphasis here is on the relation of the tools to the accounting data and how the tools are designed. The discussion is divided into four sections. The first section establishes the relationship between the tools and operations in relational algebra. The second section describes the high level implementation of the tools followed by a section describing the library routines and the approaches to common problems used in creating the tools. The final section shows how to transform relational algebra expressions into a pipeline of tools.

The variable format of event records makes the accounting data difficult to access. The quantity of accounting data requires that it be in binary format. (The rationale for these decisions are explained in Chapter 3.) These two attributes preclude analyzing the data with generic Unix tools. Any program that needs to access the data must be able to handle it in binary format and must know where each field is in each event record. If this data is hard coded into the program, then handling a change in event records or the addition of new event record types would require re-coding the program.

The Monitor system provides a solution with the schema organization, making each accounting stream self-identifying. Any new type of event record or changes to the composition of an event record is reflected in the schema. As explained in Chapter 3, the schema is produced by the sensor compiler. This does not mean that the schema and associated event records are immutable. A program can add new types of event records and alter the make up of others as long as it changes the schema accordingly. A library of common routines for manipulating the schema is provided to hide implementation details and will be discussed later in this chapter.

A conceptual viewpoint is necessary to treat any data systematically. Here, we desire to perform analysis through relational operations, but the database produced is too large to keep on disk. The concept of the

`stream` allows us to overcome the constraints on database size by accessing the tuples sequentially. The relational viewpoint requires a data abstraction, relations and tuples, and a set of relational operations. As explained in Chapter 3, the event records are treated as tuples in a relational database, mapped to the proper relation through the schema. The sensor that created an event record determines its relation, while the fields in the event record become components in a tuple. To complete this paradigm, the analysis tools provide relational operations. Additional tools are included for the administrative handling of the data. The price for this is that no relational operation that requires viewing a relation in its entirety can be performed.

7.1. Relational Database Paradigm

The view of an accounting stream as a database presents problems for specifying relational operations. Traditional databases view their data as always completely available and randomly accessible. When processing a stream, only a small portion is visible at any one time, as limited by the availability of memory and the size of the stream. Relations are therefore only visible as parts and not as a whole. Certain relational operations require dealing with two relations simultaneously. The union operation examines relations a tuple at a time, and never needs to reexamine a tuple, except possibly to handle duplicates. Ordinarily, the union of two relations would not contain duplicates, but duplicates are not well defined for event records: identical event records can be produced by operations or by identical events with insufficient granularity in the timestamp; both place the identical records adjacent to one another. If the order of the event records has not been changed, this adjacency can be used to define duplicate and duplicates can be handled without recourse to multiple passes through the data. Discarding duplicates is an option rather than the norm, since the order of the data could likely have been changed, and the timestamp is not guaranteed to be unique.

The Cartesian product and set difference operation must compare each tuple in one relation against all tuples in the other relation. Since this requires multiple passes through the stream, it is clearly impractical for large streams. The method for handling smaller streams will be described below.

The schema must reflect any alterations to the makeup of a relation. If additional processes use the stream, the schema changes must be completed and the schema written out before the tuples themselves

are processed. Relational operations on traditional databases are assumed to be non-destructive: the operations only provide new data, leaving the old data unchanged. This is not always desirable in a stream, where paring the size of the database for a given investigation can be important. Any data removed from the stream is lost to all subsequent processing, so the choice of destructive or non-destructive operations must be left to the investigator.

The identification of tuples with relations in a traditional database system is hidden from the user. In a stream, the identification is visible as `mon_cmd` and the sensor id. Any changes to a tuple is reflected in the `length` field, while a new relation requires that a new sensor id be generated.

7.2. The Tools

aggrop Unlike other analysis tools, the output is not a stream, but a table, with a line for each distinct *partition* value that contains the results of the aggregate operators on the *argument* value. The partition values come from a specified component of the target relation, while the argument value is another component of that relation. An internal data type is kept to record the aggregate value for each partition value encountered. The options are used to determine the aggregate functions to apply to the argument value, to create a format for the output. The operation can be applied to all relations made from the requisite domains and a count of the those relations not made from those domains is printed to `stderr`. The table is only printed when the end of the stream is reached, with only columns for those operations specified on the command line. The table does not have column headers so that it can be piped directly into a post-processor.

applyop Since the data structures in an accounting stream are complex, a way was needed to allow arbitrary filters use the system without having to deal with decoding the data. *Applyop* does this by projecting fields from a relation as strings to a user specified program and appending the output as new components of that relation's tuples. The command line contains the relation to affect and which of its tuple components to project to the co-routine. The result components must also be specified along with their data types, for appending to the relation. Initial processing creates a new `schema` with new `attributes` appended to the target `relation`. The processing loop of *applyop* can be considered to consist of two parts, a reader and a writer. The reader extracts the values of the projected components for the target relation and writes them as a line to the co-routine. For target relations, the writer reads a line from the co-routine and appends the result components to the tuple. The co-routine must take precautions to make sure that a result line is printed for each line it receives, otherwise the process will hang. It is acceptable to have no result components.

deschema When the schema is not wanted in the stream, or a file containing only the schema is wanted, it can be removed using *deschema*. The operation simply reads the schema, writes it to a null file, and writes the event records to the standard output.

enschema

The Accountant does not associate a schema with its event records; *enschema* is available for this purpose. *enschema* prepends the schema to the event records with a shell script using *cat*(1).

```
#!/bin/csh -f
# enschema
# $1 is schema
# $2-n are event record files
# - represents stdin
if ( -e $1 ) then
        cat $*
        exit 0
else
        echo $0 : schema file $1 not found.
        exit 1
endif
```

finitestate

The *finitestate* tool applies an instance of a finite state machine for each *partition* component value in the input. Each instance of the machine maintains in a list the `tuples` in the current *sentence*. The list is written to the output `stream` if accepted by the machine and is cleared if rejected. Either case starts a new sentence. Each accepted sentence is preceded by a relation detailing the range of timestamps in the sentence, the partition value, the name of the partition component, and the size of the sentence. This relation is added to the output schema and may be named on the command line. The partitioning component must exist in all relations in the `stream`. The value of a `tuple`'s partition component is used to determine the instance of the finite state machine to execute. When a new partition value is encountered, a new instance of the machine is allocated. Command line options allow the name of the sentence header `relation` to be changed from the default `FiniteState` and allow rejected sentences to be written out as well, but without a sentence header. The purpose of the latter option is to prevent the loss of data while still allowing sentences to be formed.

The finite state machine file consists lines of transitions in ascending order by states. The conditions on a single line represent an implicit *AND*, while additional lines for a transition represent an *OR*. The first condition satisfied determines the transition to take. The user is responsible for ensuring that the finite state machine does what is intended. The machine file is read in and parsed to create the machine itself. The parser only detects transition line syntax errors and errors in the order of states.

In addition to comparing components to immediate values in the transition line, the comparison can be the relation between a component in the current tuple and the same component in the previous tuple for in the sentence.

project

Project components from a relation or relations as an output `stream`. The input `schema` is copied to the output `schema` where it is modified to reflect the change to the relation. *Project* always places the `relation` selection fields (the structure `mon_cmd` and the field `eventnumber`) at the beginning of the description for each record, regardless of the specification on the command line. For each event record that is read, *project* creates an output record by moving the fields as specified by the output `schema` and writes it to `stdout`. Relations without projected fields may be kept or

discarded with a command line option.

relretrieve The `relretrieve` tool extracts relations from an Ingres database to form a `stream`. A complete `schema` is built from the database describing fully the relations and domains it contains. If the relations do not have the identification components, *relretrieve* assigns a unique sensor id that can be specified on the command line and a command type of `MONOP_PUTEVT_EXT` that may also be changed by a command line option.

relstore *Relstore* uses the `schema` to create relations in an Ingres database. Each `tuple` from the `stream` becomes a tuple in the corresponding relation in the Ingres database.

select *Select* parses a formula from the command line and uses it to determine the `tuples` to keep in the `stream`. A lexical analyzer is used to process the formula, that is in turn executed by a *yacc*(1) grammer. The selected `tuples` may be assigned to a new relation by command line option.

streamconvert The accounting data is stored in a binary format that can vary according to machine architecture. `Streamconvert` uses the information in the `schema`, which is not stored in a binary format, and network software to adjust the data between network and host formats as specified on the command line. Since there is no way to determine what the current format of the data is except by trying to print it, tapes should be in network format and disk files in host format. Character strings and byte fields are treated in pairs as `shorts`. Rational domains and their associated components are not portable across system architectures.

streamprint *Streamprint* uses a *libmontools* routine and the `schema` to print a `stream` in human readable format. The names for `relations` and `attributes` can be used to label the output.

tapehandle Unix does not provide a means for handling multiple reel tape files of binary data. *Tapehandle* extracts event records from multiple tape reels or creates multiple reel files from event records. The `schema` in a `stream` must be treated separately. *Tapehandle* permits the attributes of the tape and number of tapes in the file (extract only) to be specified on the command line.

union *Union* creates a single `relation` named on the command line from multiple `relations` of the same arity. Each `tuple` in the target `relations` has its sensor id changed and name changed to that of the new `relation`.

7.3. Tool Implementation

The tools share a common library of routines and a common approach to handling certain tasks. The library routines consist of those generated by IDL to handle allocation and iteration on `schema`, `relation`, and `attribute` structures, and additional routines that perform common operations on the IDL

structures and on `Mstreams` and `tuples`. These common operations consist primarily of accessing event records via the layout of the `schema`.

7.3.1. library functions

The tools use a set of common data structures. External data exists as event records, schemas, and streams. Most tools deal only with streams, leaving a few special tools to convert the records and schemas into streams and back again. Internal data from the stream exist as `databases`, `relations`, `tuples`, and `attributes` (components).

There is a subtle distinction among `attributes`, domains, and components. Domains are sets of values. A subset of the Cartesian product of a list of domains forms a relation whose members are called tuples. A tuple has a component for each domain in the list that is described by an attribute [20]. Relations, tuples, and attributes are represented directly while domains are only represented through the attributes of their associated components. If a component is to be added to a relation, then an `attribute` for it must be added to the target `relation` in the `schema`. Similarly, the traits of a component, such as its type and position in the tuple, are manipulated by altering the `attribute`. To change a component means to change a value, while to change an `attribute` means to change the characteristics of a component. The routines in the library manipulate components through their `attributes`.

object	internal structure
schema	`database`
relation	`relation`
component	`attribute`
event record	`tuple`

A suite of library routines exist (see *libmontools*(3L) manual pages in the Appendix) to manipulate `tuples`, `streams`, `relations`, and `attributes`. The IDL compiler provides additional routines in the file *schema_idl.o*, that it generates from the schema definition file, and in the IDL library, *libidl*. The IDL routines hide the implementation details of the schema, while *libmontools* hides implementation details of tuples and streams.

For each of the IDL objects `database`, `relation`, and `attribute`, *schema_idl.o* defines operations to create a new object and to operate on a list (called a *sequence*) of objects. The list operations include membership, iteration, insertion, and deletion.

The routines in *libmontools* for `streams` allow the manipulation of a stream as an I/O object. In addition to read, write, and open operations on `streams`, the `schema` may be separately processed with its own read and write.

The `tuple` manipulation routines permit accessing relations and components by name. Because IDL shares objects between sequences, whenever an object changes in one sequence, it will change in the others containing that object unless the change is made to a copy of the object. Since copying is not that simple, routines are provided to copy `schemas`, `relations`, and `attributes`. The `stream` read routine automatically sets up a tuple. If a program modifies the layout of a relation, it can use a library routine to recalculate the positions of the components. A library routine also exists to print a tuple in human readable format, with or without labels.

7.3.2. Approaches to common tasks

In addition to the library routines, the tools share common design approaches in handling arguments and options, applying operations to sets of relations, and changing IDL objects.

The analysis tools proper do not specify the input file, since they all read streams from the standard input, and, if a stream is the output, they write that stream to the standard output. Auxiliary tools, those used to convert the data, such as tapehandle, streamconvert, and enschema, have a varied choice of input and output. The arguments to the tools are relations, components, and tool specific arguments. When "–" is allowed as a relation, it signifies all relations.

Command line options can appear in any order on the command line and can be intermixed with the arguments. All options are preceded immediately by a dash ("–") or by another option. Those options that take arguments are separated from the argument by an optional space. The two most common options are to name the relation modified by the tool and to write out the relations that would ordinarily be discarded.

When a tool changes a `schema`, it must keep separate copies for the input and output `streams`. The list below summarizes the steps required.

- read the input `schema`
- create the output `schema` by copying the input `schema`
- assign the output `schema` to the output `stream`
- make any changes to the output `schema`

Note that if a new relation is added to a schema it will not interfere with existing relations and no new schema is required.

Any changes to a `relation` should only be made in the output `schema`, but a new `relation` can be added to the input `schema` without harm. To change a `relation`'s `attributes`, the IDL list manipulating routines should be used. Adding and removing `attributes` is handled by specific routines, while reordering the `attributes` requires a combination of routines. After the `relation` is complete, the positions of the components must be set by calculating the proper offset of each within the event record, making sure that alignment and byte order is maintained. A new `relation` is created by assembling a sequence of `attributes` using newly created `attributes` and the IDL routines.

Changing the value of an `attribute` is trivial for fixed length `attributes`: assign the new value to the properly cast offset into the event record. If the `attribute` is of variable length and the new value is longer, then the event record must be extended by that amount and subsequent `attributes` must be shifted. If the new length is shorter, the subsequent `attributes` can just be shifted. A new `attribute` is created by allocating space and filling it with the required values, except for the `attr_pos` field. The new `attribute` is inserted at the end of the `relation`, the positions of the components are updated, and the value for the new `attribute` can be filled in. If the `attribute` is to go somewhere other than the end of the event record, a new tuple is needed, and the components are copied in by name.

7.4. Transforming Relational Operators to Tools

The basic operations of relational algebra are listed below with the symbols used to represent them in relational algebra expressions. [20]

Operation	*Symbol*	*Results*
project	$\pi_{components}$ (*relation*)	new relation with only specified components
select	$\sigma_{formula}$ (*relation*)	new relation with tuples satisfying formula
Cartesian Product	*relation* × *relation*	new relation containing the Cartesion Product of two relations
set difference	*relation* − *relation*	relation containing tuples in first relation that aren't in second
union	*relation* ∪ *relation*	relation containing tuples in either relation

Consider the following relations

Creates(process,file) Accesses(process,file)

and relational algebra formula on these relations:

$$\pi_{process}\ (Accesses - Creates)$$

The *Creates* relation contains tuples whose components are pairs consisting of a file and the process that created it. The *Accesses* relation contains pairs of a file and a process that either read from or wrote to the file. The expression creates a relation containing all those processes that only accessed files but never created any. The next section will detail how to transform this expression into a series of tool commands, and on transforming any expression into tool commands.

Ordinarily, expressions containing set difference and Cartesian product would be evaluated inside a conventional relational database system though the tools *relstore* and *relretrieve*. If, after reducing the stream as much as possible it still can't fit on a disk, then these operations can be simulated to a limited extent using the analysis tools. To perform this expression with the analysis tools requires rewriting the formula into a form the tools can manage, possibly requiring more than one pass through the data. For large streams this is still prohibitively expensive, but sometimes it is possible to reduce the stream significantly and to perform the evaluation with only a single pass. Each part of the expression will be dealt with in turn.

Notice that the set difference operation is used in coordination with the project operation, where it forms the relational algebra equivalent of a loop. The stream must effectively become two streams, each consisting of only one relation. The set difference is performed by checking each tuple in one relation for membership in the other relation. The shell doesn't permit any way of specifying two streams, so one stream must be turned into a file. This file can then be used by the following *awk* program and applied to

the stream by *applyop*.

```
FILENAME == "Creators" {
        creators[$1$2] = 1;
        next
}
creators[$1$2]==1 {
        print 0
        next
}
{
        print 1
}
```

Awk operates on lines of character data, breaking the lines into space separated fields that are numbered `$1` through `$9`. The code is broken into blocks with conditional statements that *awk* uses to determine which blocks to execute. Execution continues for each block whose condition is met, unless processing for the line is explicitly stopped.

The first block in this program is selected when the input file is `Creators` and builds an associative array based on the catenation of the first two fields (indicated as `$1` and `$2`) as the index into the array. Each entry is set to `1` to represent the presence of the catenation value. Further processing of the line is bypassed by the **`next`** keyword. The second and third blocks of code are executed on the remainder of the input. These print a `0` if the current line has an entry in the associative array and a `1` if no entry is found.

The *Creators* file is extracted from the stream by the pipeline:

```
% project Creates process file < stream | streamprint -uk > Creators
```

Project extracts the tuples from the stream and *streamprint* converts the tuples to human and *awk* readable format, putting the results in *Creators*. *Applyop* takes the *awk* program, the *Creators* file, and the input stream and creates a file in stream format called *Accessers*.

```
% applyop awk -p '-f awkfile Creators -' Accesses process file -
accept:boolean < stream | select Accesses "accept=true" > Accessers
```

Applyop projects the values of the *process* and *file* components of the relation *Accesses* in a character format to the *awk* program and appends a boolean component *accept* (*true* and *false* are represented as `1` and `0`), built from the *awk* program's output. *Select* uses the new component to create *Accessers*, that

contains tuples of those processes that accessed at least one file that they didn't create. Since the associative array `creators` is kept in main memory, its size is limited. This approach is effective because most domains in a stream have values in a limited range.

The Cartesian product of two relations is treated similarly to the set difference but is distinguished from set difference in that the argument relations need not be of the same arity, nor have similarly named components. As with set difference, Cartesian product is used predominantly with select and project, but an additional difficulty arises owing to the size of the intermediate relation. While set difference splits the stream, thus reducing the size of the database, Cartesian product greatly increases the size of the database. Each tuple in the new relation contains as many components as the two argument relations combined, and the number of tuples in the new relation is the product of the number in the two arguments. If the entire Cartesian product is required throughout an analysis session, the investigation will be limited by the amount of memory available in the system. To save space, the relations and components not taking part in the expression should be removed from the stream as early as possible through *select* and *project*. By decomposing the expression, more space can be saved, at the cost of additional passes through the data. Sufficiently small streams, where the intermediate results can fit on disk, can utilize *relstore* and *relretrieve*, while sufficiently large streams make more than a single pass through the data prohibitive. Consider the relations *Creates* and *Accesses* again, with an additional component to *Accesses* that records the number of bytes accessed.

Creates(process,file) Accesses(process,file,accesscount)

Again we wish to deal only with processes that don't access files they create, with an extra condition that the access be large enough, 500 bytes, for example. Since the arity of the two relations differ, set difference is not defined, but an equivalent result can be obtained by combining project, select, and Cartesian product.

$$\pi_{process2}\left[\sigma_{process1 \neq process2 \wedge file1 = file2 \wedge accesscount > 500}\left(Creates \times Accesses\right)\right]$$

Despite the change to *Accesses*, the same pipelines as used in the set difference example above can be used to satisfy the Cartesian product expression with only a change to the selection formula:

```
select Accesses "accept = true & accesscount > 500"
```

This works because the select formula in this expression is similar in effect to the set difference expression in the previous example, having only a single additional criteria. Differences in arity are unimportant to *applyop*, since an implied project occurs before applying the co-routine. While this is a fortuitous choice, it is not unlike many applications of Cartesian product.

More complex uses of Cartesian product that can't be simulated by iteration cannot be handled by the tools, since any stream small enough to permit multiple passes can be better served by using a conventional database system.

The applications that use a conventional database can be far more general than for a stream database. At Carnegie-Mellon University's Computer Science Department, using the Monitor and a preliminary version of the Accountant, accounting data has been placed into a relational database that can be viewed using a graphical representation of an accounting session [26]. A given accounting session is limited in duration and in the number of processes run, because of space and processing time considerations. Several sessions are kept in the database, and each is viewed separately. The graphical view of the session presents each process as a time line, with event records appearing as bars on that line. Operations on the view include panning through time, and zooming in on a moment, up to viewing the contents of an event record.

Intermediate results in processing a relational algebra query require the ability to create new relations. Tools that modify a stream provide this capability by allowing their target relation to be renamed. This is only necessary for mnemonic reasons are when the target relation and the new relation must co-exist. If a user program needs to create new relations or modify existing relations, then it must modify the schema to contain the new relations and pass the new schema to the output before writing any tuples to the output. Details of what must be modified will be described below.

There are occasions when the investigation of a stream requires dealing with collections of tuples from a variety of relations. While this is done in a traditional database system through the Cartesian product, we have already seen that this is difficult to do with streams, and impossible if the stream is only to be processed once. The order of tuples in an unmodified stream represents the order that the event records were created. This differs from the traditional arrangement where the order of tuples is not significant. The

tool *finitestate* (see the manual pages in the Appendix) provides a means for examining series of tuples, and deciding acceptance or rejection based on a finite state machine provided by the user. A separate instance of the finite state machine is created for each partition value in the stream. The state of the machine represents the memory of what has happened for that partition value. As each new tuple is entered, it is stored in the *sentence* for its partition value. When an accepting state is reached, the sentence is written out, preceded by a header tuple containing information about the sentence. If a rejecting state is reached, the accumulated sentence is discarded.

Finitestate was designed to enable the study of the pathname locality when opening files. When a process opens a file, it specifies the path the kernel must traverse to find that file. This is monitored by the Name-Start, NextComponent, and OpenSuccessful sensors (see Chapter 4). The sequence of the tuples created by these sensors marks the path taken by the kernel to find the file. The length of the chain of tuples shows the proximity of the file to the current working directory. The relationship between the members of the chain is shown only by their relative positions in the stream, and the absence of any tuples with the same initiator within the stream. This preclude analysis by a relational algebra expression. *Finitestate* can extract the chains as separate sentences, and, through a sentence's header tuple, provide a hook that can be caught by relational algebra.

The analysis tools do not provide a complete set of relational algebra operations, but they do provide most of the common operations and the ability to use a complete relational algebra through the tools *relstore* and *relretrieve*. The tools directly execute union, project, and select, and, using combinations of tools, set difference. The Cartesian product is only available through Ingres, but can be simulated under certain circumstances. The ability to create new relations is provided in the tools, and the additional ability to examine sets of dissimilar relations through *finitestate* and *applyop*. Unlike traditional databases, the tools have the capability of handling arbitrarily large amounts of data.

Chapter 8

Conclusion and Future Work

The system as currently implemented consists of sensors and schemas generated by hand from sensor descriptions, the Monitor system call to manage data and sensors, the Accountant to control the accounting process through the Monitor, and and some of the suite of analysis tools to treat the data.

8.1. Implementation

The use of automatic code generation for schemas reduced the time needed for developing the analysis tools and the complexity of those tools. Schemas provide a means to self-define a stream, so that generic tools do not have to have hard coded knowledge of the event records. When the sensor compiler is completed, new sensors can be defined in terms of their target parameters, with their schemas generated automatically, leaving only placement and parameter decisions to the user.

Despite the code generating tools, one tool was still difficult to develop. The interaction of multiple processes and the lack of existing tools for specifying the interaction made the program *applyop* difficult to write. Existing debugging aids cannot handle multiple processes well, and, while Unix does have interprocess communication facilities, these are really designed to work between programs that expect such communication, rather than between arbitrary programs.

Simulating the kernel, Accountant, and Monitor allowed much of the system to be debugged before proceeding to the target environment. This was especially important with the kernel routines, which otherwise required large amounts of overhead for changes. Changes to parts of the system could be tested quickly using the simulator before undergoing the expense of installing a new kernel. Development in an environment where crashes and performance were isolated (a workstation) reduced the remaining testing time on a real system.

Distributing the system to other sites permitted independent testing, and caught problems that would have otherwise gone unnoticed. Users at the distribution sites operated in a slightly different environment and under a different set of assumptions leading to the detection of errors that were missed in testing and the detection of dependencies on local system modifications. The work at other sites effectively trebled the testing, as well as contributing important ideas to improving the system.

While the complexity of an operating system makes proper placement of sensors difficult, stateless distributed systems are even more difficult to instrument. As mentioned in Chapter 4, the act of obtaining information can cause the sensors to recurse infinitely. This will be discussed further below.

8.2. Operation and Analysis

The monitoring system is easy to use to collect file system data, and by extension, general data from the kernel. For this purpose, the reduced Accountant *acct* suffices to control the system. Placing the sensors in the kernel allows monitoring file system usage across all processes in the system and requires no changes to user programs. Individual processes can be extracted from the data stream and analyzed independently, or the system activity can be viewed on the whole. Sensors can be added to programs to create a unified set of data containing both user and kernel information, though this ability has not been fully demonstrated.

The relation database approach, while flexible, has weaknesses when handling arbitrary amounts of data. Cartesian product and set difference are impossible in a pure form (a pipeline can't store an arbitrary amount of data), and union must be slightly redefined to be implemented. Using a set of analysis filters is a good approach: it is flexible, allows handling of cumbersome amounts of data, and is extendible, especially through the library routines. Of the tools, the ones not directly related to relational operations are among the most useful: *applyop* and *finitestate*. The utility of *applyop* is in allowing arbitrary Unix filters to be used on stream data, while *finitestate* allows handling data that is only represented implicitly in the stream. The order of the event records, an implicit datum, is altered by *finitestate*, but can be recovered by using *applyop* to append the position explicitly to each record.

8.3. Future Work

This section discusses work remaining to complete the existing parts of the system, identifies the systems shortcomings, and suggests extensions. Completing the system requires refining the communication between the data gathering components of the system, completing the analysis tools and the sensor compiler, and measuring the overhead generated by monitoring.

The communication between the Accountant and the Monitor is now initiated by a timer. Signals are a better way of initiating Accountant/Monitor interaction, but add complexity to the development and testing, since simulating the kernel would require a separate process. When new features are added, it would be better to be able to disable signaling and return to timaing during the testing process. Signals and interprocess communication will also be required to implement an interactive controller for the Accountant and user process/Accountant interaction. Some of this code already exists, but requires upgrading and testing.

The monitoring system is designed to be general, but to be truly useful for arbitrary monitoring, the automatic generation of sensors and schemas from a description is needed. The sensor generation should be straightforward, since the methods for converting sensor descriptions to code have already been developed. The use of IDL should ease the schema generation part of the compilation.

Most of the analysis tools have been implemented. Of those described but not implemented are *relstore*, *relretrieve*, *streamconvert*, and *tapehandle*. These are not necessary for many operations, but are needed for transferring data between machines of different types and for using Ingres.

There are shortcomings in the design and implementation. Viewing the data produced by the sensors currently placed in the kernel suggests some additional information that would be useful. If a single process is to be studied, it is now impossible to determine if the entire activity of the process is covered in the data or whether accounting was present through only a portion of the process's execution. It is also impossible to know when all event records produced for a process have been seen. The addition of sensors to record the beginning and ending of processes would enable both of these to be determined. Sensors of this nature are being used at C-MU to depict processes graphically along a timeline, so that at least one implementation of these sensors exists [26].

In its current state, the monitoring system produces enormous amounts of data, and itself requires monitoring to ensure that the data doesn't overload the system. Unix file system usage makes for many essentially identical event records, allowing space to be saved by using ibid and count sensors.

As mentioned in chapters 4 and 5, efforts are being made to run the system on a multiprocessor. Handling the resulting concurrency requires a genuine semaphore in both the kernel sensors and the Monitor. Since this is not available directly in Unix 4.2BSD, assembly language routines will be necessary (it is assumed that a multiprocessor will have the required operations).

Approaches to handling distributed file systems should also be investigated. Design and position of the sensors become even more critical here, because of the potentially high overhead and possibility of sensor recursion. Since the Sun Microsystems' Network File System (NFS) is a stateless system [25], file information must be looked up each time to insure its accuracy. This can't be done with the current sensors, but requires designing new sensors for each part of NFS. While the NFS requires a different set of sensors, it does not require a different Monitor or Accountant.

Bibliography

1. Deitel, Harvey M. *An Introduction to Operating Systems*. Addison-Wesley, (1984).

2. Ferrari, D., Spadoni, M. *Experimental Computer Performance Evaluation*. Elsevier North-Holland, Inc., New York, NY, (1980).

3. Ferrari, D., G., Serazzi, A., and Zeigner *Measurement and Tuning of Computer Systems*. Prentice-Hall, Inc., Englewood Cliffs, NJ, (1983).

4. Floyd, Rick Short-Term File Reference Patterns in a UNIX Environment. TR 177, Computer Science Department, University of Rochester, (March 1986).

5. Garcia-Molina, H, Germano, Jr., F, Kohler, W.H. Debugging a Distributed Computing System. *IEEE Transactions on Software Engineering SE-10*, 2 (March 1984), 210-219.

6. Godfrey, M. D., Hendry, D. F., Hermans, H. J., Hessenberg, R. K. *Machine-Independent Organic Software Tools (MINT)*. Academic Press, New York, NY, (1982).

7. Goldberg, A., Popek, G. Measurements of a Distributed Operating System: LOCUS. ucla, (1982).

8. Graham, S. L., Kessler, P. B., McKusick, M. K. gprof: a Call Graph Execution Profiler. in Proceedings of the SIGPlan '82 Symposium on Compiler Construction, ACM, Boston, MA, (June 1982), 120-126.

9. Kupfer, M. Performance of a Remote Instrumentation Program. UCB/CSD 85/223, Computer Science Division (EECS), University of California, Berkeley, (February 1985).

10. Kupfer, M.D. An Appraisal of the Instrumentation in Berkeley UNIX 4.2BSD. PROGRES ReportUCB/CSD 85/246, University of California, (June 1985).

11. Miller, B.P., Macrander, C., and Sechrest, S. A Distributed Programs Monitor for Berkeley UNIX. UCB/CSD 84/206, Computer Science Division (EECS), University of California, Berkeley, (October 1984).

12. Ousterhout, J.K, Da Costa, H., Harrison, D., Kunze, J.A., Kupfer, M., Thompson, J.G. A Trace-Driven Analysis of the UNIX 4.2BSD File System. UCB/CSD 85/230, University of California, (April 1985).

13. Satyanarayanan, M. A Study of File Sizes and Functional Lifetimes. in Proceedings of the Eigth Symposium on Operating System Principles, Asilomar, CA, (December 1981).

14. Satyanarayanan, M. fscript1 - Benchmark suite for the Unix file system. magnetic tape

15. Smith, A.J. Analysis of Long Term File Reference Patterns for Application to File Migration Algorithms. *IEEE Transactions on Software Engineering SE-7*, 4 (July 1981), 403-417.

16. Snodgrass, R. Monitoring Distributed Systems: A Relational Approach. Ph.D. Dissertation, Computer Science Department, Carnegie-Mellon University, (December 1982).

17. Snodgrass, R. The Temporal Query Language TQuel. in Proceedings of the Third ACM SIGAct-SIGMOD Symposium on Principles of Database Systems, Waterloo, Ontario, Canada, (April 1984), 204-212.

18. Stritter, E.P. File Migration. Ph.D. Dissertation, Stanford University, (March 1977).

19. Thompson, K. UNIX Implementation. *The Bell System Technical Journal 57*, 6, part 2 (July/August 1978), 15.

20. Ullman, J.D. *Principles of Database Systems, Second Edition* (Computer Software Engineering Series). Computer Science Press, 11 Taft Court, Rockville, Maryland 20850, (1982).

21. Weinberger, P.J. Cheap Dynamic Instruction Counting. *AT&T Bell Laboratories Technical Journal 63*, 8, part 2 (October 1984), 1815-1826.

22. Measuring, Modelling and Evaluating Computer Systems. Beilner, H., Gelenbe, E. (Ed.), North-Holland Publishing Co., Amsterdam, (1977).

23. Performance 81. Kylstra, F. J. (Ed.), Elsevier North-Holland, Inc., New York, NY, (1981).

24. Software Metrics: An Analysis and Evaluation. Perlis, A., Sayward, F., Shaw, M. (Ed.), The MIT Press, Cambridge, MA, (1981).

25. Networking on the Sun Workstation (2A). Sun Microsystems, Inc, 2550 Garcia Avenue, Mountain View, CA 94043, (1985).

26. Site demonstration of graphics interface at C-MU.

Appendix A

Generating Standard Sensors

This is SoftLab internal document 8, explaining the generation of sensor macros from a sensor description.

Abstract

Directions for compiling standard sensors for both kernel and user processes from a sensor description. Includes examples of sensor descriptions and the sensors that should be generated.

This document describes the structure of sensors to be used with the UNIX™ operating system. The source description for a given sensor will be in a Sensor Descriptor Language (described in a separate document). The generated code is placed into a file to be *included* in the target routines. Each sensor is a C macro with appropriate parameters determined by analyzing the source description. For every sensor described there will be two macros generated: the macro that constitutes the sensor proper and another macro that enables and disables the sensor. Each sensor description will also generate an ascii print format. This will be used to display the data from that sensor.

The code in this document is displayed in a constant width font with the keywords enboldened. Italics are used for comments in the code and for portions of the code that vary from one run of the sensor translator to another. Technical terms in the text are italicized when they first occur.

1. Enabling and disabling sensors

The sensors are numbered sequentially, starting at one. This number is called the *event number*. Each sensor is associated with one or more unique bits in the `mon_enablevector` array (see below), which is effectively an array of boolean bit fields arranged in short integers. The status of the bits corresponding to a particular sensor determine the execution of the sensor. One bit is assigned to each standard sensor. The event number divided by 16 yields the *offset*, the index of the short integer which contains the bit, and the sensor number mod 16 yields the *mask*, which represents the power of two which selects the appropriate bit, both of which are known at compile time. If the short integer selected by the *offset* is *OR* ed with the *mask*, then the bit will be set and the sensor enabled. If it is *AND* ed with the complement of *mask*, then the sensor will be disabled.

1.1. Enabling and disabling user sensors

An array `monu__boolvec` is declared at the head of the user sensor definitions. As with the kernel sensors, this is a boolean array of short integers. Sensors are enabled and disabled by manipulating this array. Note that this array is global to the user program but is not directly accessible by the kernel. The cumbersome names used in the user sensors are necessary to avoid conflict with variable names in the user program.

2. Sensor Generation

This section describes the include file containing the macro definitions for the sensors. There is one include file generated for the entire operating system.

2.1. Include file header

The following definitions are placed at the beginning of the include file for operating system sensors.

```
#ifdef   KERNEL
#include "../monitor/mondefs.h"
#include "../monitor/montypes.h"
#ifndef  ntohs
#include "../netinet/in.h"
#endif
#else
#include <monitor/mondefs.h>
#include <monitor/montypes.h>
```

```
#ifndef  ntohs
#include <netinet/in.h>
#endif
#endif
#define  Timestamp (long)((time.tv_sec << 15) | (time.tv_usec >> 5))
extern int            mon_semaphore;
extern unsigned char  *mon_write_ptr;
extern unsigned char  *mon_read_ptr;
extern unsigned char  *mon_eventvector_end;
extern int            mon_eventvector_count;
extern int            mon_oflow_count;
extern unsigned short mon_enablevector[];
extern unsigned char  *mon_wraparound();
```

The conditional compilation is present for use when the machine is run in *standalone* mode, when most of the file system is unavailable. The *include* files `montypes.h` and `mondefs.h` contain the C type definitions and declarations used throughout the monitoring system. The time stamp is defined as a macro to enable easier modification to the granularity. It has a period of 2^{17} seconds, with a resolution of approximately $1/2^{15}$ seconds (a tick is approximately 30.5 microseconds). The low order bits of the `time.tv_usec` field are dropped off because the system clock isn't sufficiently fast to make them reliable. The variables are as follows: `mon_eventvector` points to the low address of the ring buffer, `mon_write_ptr` is the tail and `mon_read_ptr` is the head, `mon_eventvector_count` is the amount of data filled, and `mon_eventvector_end` points to the nominal high address. The variable `mon_oflow_count` is incremented when data has been lost due to a full buffer; `mon_enablevector` is an array used to indicate whether a sensor is active; and `mon_wraparound` handles the wrap around condition for the ring buffer.

2.1.1. Include file header for user sensors

The header for user sensors is much simpler, consisting of the array of status bits and an event buffer capable of holding a single event, and some *include* files that hold need declarations and definitions. The need for the remaining lines, those referring to ring buffer management, is obviated by using a system call to handle the actual storage of the event records.

```
#include <monitor/montypes.h>
#include <monitor/mondefs.h>
#include <netinet/in.h>
#include <sys/syslocal.h>
short        monu__boolvec[16];
mon_putevent monu__sbuffer;
```

2.2. Sensor Definition

The code inside the sensor proper consists of conditional compilation statements, a control section and the logic to store the sensor information in the ring buffer.

2.2.1. Conditional compilation

The conditional compilation statements are used to permit removal of the sensors from the operating system without having to individually delete each section of code. Bracketing each sensor definition will be

```
#ifdef  MONITOR
#define sensorname (parameter1, ...., parameterN)  \
        .                                          \
        .                                          \
        .                                          \
#else
```

```
#define sensorname (parameter1, ..., parameterx)
#endif  MONITOR
```

All the sensors can thus be installed by defining MONITOR when compiled. The flag -DMONITOR is specified to the compiler to install the sensors; its absence removes them. Note that most sensors are initially disabled when installed, so that use of this flag will *not* guarantee that event records are generated. However, it does imply that the enable bit(s) will be checked each time the sensor is encountered.

2.2.2. Control section of sensor

The control section determines whether the sensor is enabled, whether there is sufficient room in the ring buffer to hold it, and if the ring buffer must wrap around. Kernel sensors also contain code that checks for concurrency with other kernel sensors and prevents execution if concurrency is detected. If there is insufficient room to hold the event record then the sensor code is bypassed and mon_oflow_count The ring buffer is constructed with an *appendix* of data locations which is used for an overflow area so that only one check per sensor is required to deal with wrap around. If the buffer has wrapped around, then the data from the nominal end of the ring buffer to the current position of the write head (which is now in the appendix) is copied to the front of the buffer by the subroutine mon_wraparound in the module local_syscalls.c. We know that there is room at the front of the ring buffer since that compare has already been done. The generated code for controlling sensor execution is:

```
if (*(mon_enablevector + offset) & mask)                                    \
{                                                                           \
    if (mon_semaphore++ == 0)                                               \
    {                                                                       \
        if (mon_eventvector_count <                                         \
            MON_EVENTVECSIZE + length_of_event_record*2 - sizeof(mon_errrec))  \
        {                                                                   \
            register mon_putevent *reg_ptr = (mon_putevent *)mon_write_ptr; \
            register short *sen_fields = reg_ptr->fields;                   \

            Body of Sensor, to be explained below

            if ( sen_fields > (short *)mon_eventvector_end)                 \
                mon_write_ptr = mon_wraparound((unsigned char *)sen_fields); \
            else                                                            \
                mon_write_ptr = (unsigned char *)sen_fields;                \
        }                                                                   \
        else mon_oflow_count++;                                             \
        mon_semaphore--;                                                    \
    }                                                                       \
}
```

offset, *mask*, *length_of_event_record*, and MON_EVENTVECSIZE are constants known at compile time. The *offset* is the index of the short integer in mon_enablevector described above, containing the bit that controls this sensor. The *mask* has a 1 at the appropriate bit location for this sensor and zeros for the other positions. The correlation between the event number and the offset and mask values was discussed in Section 1.

The variable mon_semaphore is used to detect concurrency among the kernel sensors. It is compared to zero and incremented before the main body of the sensor and decremented after it. If it is found to be non-zero in the test, the main body of the sensor is bypassed and mon_semaphore is not decremented, effectively disabling all kernel sensors. This is necessary to prevent corruption of the pointers shared by the sensors, which could cause the operating system to crash.

The test to determine if there is room in the ring buffer uses the variable mon_eventvector_count and the constants MON_EVENTVECSIZE, sizeof(mon_errrec), and *length_of_event_record*. The variable mon_eventvector_count contains the number of bytes currently filled in the ring buffer. Since *length_of_event_record* is in short integers, it is multiplied by two

to yield the length in chars. sizeof(mon_errrec) is also subtracted as room for an error record. The size of the ring buffer is determined from the constant MON_EVENTVECSIZE For sensors with fixed length event records, the actual lengths of their event records are known at compile time. Sensors with variable length event records use the maximum length for the comparison, though the actual length is later inserted into the event record. Note that each line in the macro definition ends with a back slash except for the final line.

2.3. Storage of sensor information

The logic to store the sensor information consists of an initialization section and a parameter storage section. User sensors and operating system sensors differ only in their initialization. Operating system sensors will be covered first in detail, followed by the differences for user sensors.

2.3.1. Initialization

Initialization for a sensor begins with loading the address of the first open position in the ring buffer into a register pointer of type mon_putevent which is duplicated here from the include file montypes.h

```
struct mon_pevt {
        struct mon_cmd {
              unsigned char type;
              unsigned char length;
              } cmd;
        short eventnumber;
        short performer;
        long  object;
        short initiator;
        long  timestamp;
        short fields[EVENT_LIMIT];
        };
typedef struct mon_pevt mon_putevent;
```

If there are no character strings among the sensor parameters, the type of event, MONOP_PUTEVENT_INT and the length in short integers are moved into the structure mon_cmd at the beginning of mon_putevent. The length is multiplied by two to yield the length in chars and is added to mon_eventvector_count. The event number of the sensor is loaded into the eventnumber field. See Section 1 for the discussion of how this is determined. The performer is set to zero and bypassed. It is used to record the *process id*, which for a kernel sensor is 0. The id of the sensor's object is loaded into the object field. The initiator field is not used for kernel sensors. If a time stamp is required, it is moved into the timestamp field, which is otherwise set to zero.

```
reg_ptr->cmd.type      = MONOP_PUTEVENT_INT; /* when no strings */ \
reg_ptr->cmd.length    = length;                                  \
mon_eventvector_count += length * 2;                              \
reg_ptr->performer     &= 0;                  /* for process id  */ \
reg_ptr->eventnumber   = event number;                            \
reg_ptr->object        = id of sensor's object;                   \
reg_ptr->timestamp     = Timestamp;           /* when required   */ \

/* Fill in sensor specific fields – described in 2.3.2 */
```

If strings are present the length of the event record can only be determined after the character string's length has been determined. Once the record's length has been determined it can be loaded into the buffer. The length is calculated by subtracting the initial position of the ring buffer pointer from the position after the string is loaded.

```
register mon_string sen_f_ptr = (mon_string)string_parameter;   \
```

```
register mon_string sen_f_end = (sen_f_ptr + max_length/sizeof(*mon_string)); \
register short sen_length;                                                    \
register short *sen_fields = reg_ptr->fields;                                 \
reg_ptr->performer    &= 0;                                                   \
reg_ptr->eventnumber  = event number ;                                        \
reg_ptr->object       = id of sensor's object ;                               \
reg_ptr->timestamp    = Timestamp;                                            \

/* Fill in sensor specific fields – described in 2.3.2 */

sen_length            = sen_fields - (short *)reg_ptr;                        \
*reg_ptr->cmd.type    = command ;                                             \
*reg_ptr->cmd.length  = sen_length;                                           \
mon_eventvector_count += sen_length * 2;                                      \
```

Subtracting the beginning position in the ring buffer from the current position yields the length of the event record. The length is placed in `cmd.length` in the event record. The length is then converted to bytes and added to `mon_eventvector_count`.

2.3.2. Parameter Storage

The parameters of the sensor are filled in at successive positions past the beginning of *fields*. There are three cases: two byte integer, four byte double integer, and character strings. For a two byte integer, the parameter is moved to the next open position.

```
*(sen_fields++) = parameter; \
```

For a double integer, the parameter is also moved to the next open position, but `reg_ptr` must be incremented by two. Since the increment operator (++) will only increment by one, a separate addition is required.

```
*(long *)(sen_fields) = parameter; \
sen_fields += 2;                   \
```

To insert character strings into the buffer, the pointers must be set to the first element in the string and to the last element that is desired. This must be done at the beginning of the sensor's block. These are then used to step through the string.

```
register mon_string sen_f_ptr = (mon_string)string_parameter; \
register mon_string sen_f_end = (sen_f_ptr + desired_length); \
```

desired_length is the maximum length allowed for character string parameters, in units of `sizeof(*mon_string)`. This can be calculated by taking the desired length in characters and dividing by `sizeof(*mon_string)`. The use of `mon_string` will be described below. The compiler will resolve this term into a constant. When a parameter is known to have a maximum length shorter than that for all sensors, or only a shorter length is desired for this particular parameter, that length should be used instead. This is known at compile time. Subsequent string parameters reinitialize the same pointers.

The event buffer consists of short integers. Many machines do not permit the assignment of short integer pointers to arbitrary boundaries, but are likely to permit such for character strings. The sensor code handles this by a C `typedef` and two macros whose definitions vary depending upon this characteristic.

```
#ifdef SHORTALIGN
typedef char      *mon_string;
#else
typedef short     *mon_string;
#endif
```

```
#ifdef   SHORTALIGN                  /* For fetches that may need alignment */
#define  PackStr(ptr) ( ntohs((*ptr<<8) | (*(ptr+1))) )
#define  NotEOS(ptr,last) (ptr <= last && *ptr++&0xff && *ptr++&0xff)
#else
#define  PackStr(ptr) (*ptr)
#define  NotEOS(ptr,last) (ptr <= last && *ptr&0x00ff && *ptr++&0xff00)
#endif
```

The macro `PackStr` groups the characters of the string in pairs, either by using short integers or by shifting characters, while `NotEOS` determines when the end of the string is reached (a null character) or the maximum length allowed is reached. The system macro `ntohs` is used to prevent byte swapping in `PackStr` but is not necessary for the comparisons in the **while** statement, since both possibilities are checked.

The variable length of character strings causes the lengths of event records that contain them to be variable. The maximum string length must therefore be used in determining whether there is room remaining in the ring buffer. It also requires those steps mentioned above under initialization to insure that the actual length of the event record is entered into the ring buffer. The macro `NotEOS` handles the termination condition in the **while** statement. Inside the loop, the string is moved two characters at a time into the address pointed at by the register pointer:

```
do { *(sen_fields++) = Pack(sen_f_ptr); }    \
    while ( NotEOS(sen_f_ptr, sen_f_end) ); \
*(sen_fields - 1) &= ntohs(0xff00);          \
```

The string handling assumes all strings terminate in binary zero. The last byte of the string in the ring buffer is ANDed with 0xff00 to allow for truncating a string. This must be passed to `ntohs` so that the action occurs properly, regardless of the byte order of the host machine. Truncation results when the string is longer than that allowed, or the sensor definition specifies that only a certain length is required and the parameter exceeds this.

2.3.3. Storage for user sensors

The data fields of a user sensor are filled using the same general code as the operating system sensors. The major differences are that user sensors need not handle wraparound conditions or concurrency, utilize different naming conventions, and use a system call to write into the ring buffer.

User sensors load the address of a buffer of type `mon_putevent` into a register, where data is stored prior to being transferred to the ring buffer by a system call. This buffer is global to the program and is used by all sensors in the program.

```
register mon_putevent *monu__reg_buf = &monu__sbuffer;        \
register short        *u_sen_fields = (short *)monu__reg_buf; \
```

The struct `mon_cmd` is filled in, using `MONOP_PUTEVENT_EXT` for `type`. `Length` is treated identically to operating system sensors, as are `eventnumber` and `object`. `Performer` is ignored. It will be filled in with the caller's *pid* by the system call used to store the sensor. The `timestamp` field is filled using the same system variables as for the kernel sensors.

```
monu__reg_buf->eventnumber = MONOP_PUTEVENT_EXT; \
monu__reg_buf->object      = object;             \
monu__reg_buf->timestamp   = timestamp;          \
```

The length of variable length records is determined through the use of the variables `u_sen_f_ptr` `u_sen_f_end` and `u_reg_length` and the same algorithms used by operating system sensors.

```
register mon_string u_sen_f_ptr = ( mon_string )str_parm_1; \
```

```
register mon_string u_sen_f_end = u_sen_f_ptr + desired_length; \
register short      u_reg_length;                               \
```

The algorithms used by the operating system sensors are used to fill `monu__sbuffer` This does not store the data, however. A system call to a monitor in the kernel is required to write into the ring buffer.

```
syscall(SYSLOCAL, MONITOR, (unsigned char *)monu__sbuffer);
```

SYSLOCAL and MONITOR are defined in a header file `/sys/h/syslocal.h`

3. Examples of Code Generated from Sensor Descriptor File Below are the definition of a user sensor followed by the generated code and the definitions of two kernel sensors followed by their generated code.

3.1. A User Sensor Definition

```
Event UserXamplSensor (obj: integer object;
                       str_parm_1: string[127];
                       sh_parm_2:  integer;
                       str_parm_3: string[127];
                       lg_parm_4:  doubleinteger) is
      timestamped, sensortraced;
```

3.2. An Include File Containing A Single User–Defined Sensor

```
#include <monitor/montypes.h>
#include <monitor/mondefs.h>
#include <netinet/in.h>
#include <sys/syslocal.h>
short monu__boolvec[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
mon_putevent monu__sbuffer;
#define  UserXamplSensor(obj, str_parm_1, sh_parm_2, str_parm_3, lg_parm_4)     \
if  ( monu__boolvec[0] & 0x1 )                                                  \
{                                                                               \
     register mon_putevent  *monu__reg_buf = &monu__sbuffer;                    \
     register short  *u_sen_fields = ( short * )monu__reg_buf;                  \
     register mon_string  u_sen_f_ptr = ( mon_string )str_parm_1;               \
     register mon_string  u_sen_f_end = u_sen_f_ptr + 127/sizeof(*mon_string);  \
     register short  u_reg_length;                                              \
     monu__reg_buf->eventnumber = 1;                                            \
     monu__reg_buf->object = obj;                                               \
     monu__reg_buf->timestamp = 1;                                              \
     do { *u_sen_fields++ = Pack(u_sen_f_ptr); }                                \
     while  ( NotEOS(u_sen_f_ptr, u_sen_f_end) );                               \
     *(u_sen_fields - 1) &= ntohs(0xff00);                                      \
     *u_sen_fields++ = sh_parm_2;                                               \
     u_sen_f_ptr = ( mon_string  )str_parm_3;                                   \
     u_sen_f_end = u_sen_f_ptr + 127/sizeof(*mon_string);                       \
     do { *u_sen_fields++ = Pack(u_sen_f_ptr); }                                \
     while  ( NotEOS(u_sen_f_ptr, u_sen_f_end) );                               \
     *u_sen_fields &= ntohs(0xff00);                                            \
     *( long * )u_sen_fields = lg_parm_4;                                       \
     u_sen_fields += 2;                                                         \
     u_reg_length = u_sen_fields - (short *)monu__reg_buff;                     \
     monu__reg_buff->cmd.type = MONOP_PUTEVENT_EXT;                             \
     monu__reg_buff->cmd.length = u_reg_length;                                 \
```

```
        syscall(SYSLOCAL, MONITOR, ( unsigned char * )&monu__sbuffer);          \
}
```

3.3. Kernel Sensor Definitions

```
Event ReadSensor (device, inumber: integer object;
            filepos: doubleinteger;
            actualcount: integer) is
      timestamped, sensortraced;
Event NextComponent (device, inumber: integer object;
               filename: string[127]) is
      timestamped, sensortraced;
```

3.4. An Include File Containing Two Kernel Sensors

```
#ifdef KERNEL
#include "../monitor/mondefs.h"
#include "../monitor/montypes.h"
#ifndef ntohs
#include "../netinet/in.h"
#endif
#else
#include <monitor/mondefs.h>
#include <monitor/montypes.h>
#ifndef ntohs
#include <netinet/in.h>
#endif
#endif
#define Timestamp (long)((time.tv_sec << 15) | (time.tv_usec >> 5))
extern int mon_semaphore;
extern unsigned char *mon_write_ptr;
extern unsigned char *mon_read_ptr;
extern unsigned char *mon_eventvector_end;
extern int     mon_eventvector_count;
extern int     mon_oflow_count;
extern unsigned short mon_enablevector[];
extern unsigned char *mon_wraparound();
#ifdef MONITOR
#define ReadSensor(device,inumber,filepos,actualcount)                          \
if ( *(mon_enablevector+0) & 1<<8 )                                             \
{                                                                               \
    if (mon_semaphore++ == 0)                                                   \
    {                                                                           \
        if (mon_eventvector_count <                                             \
            MON_EVENTVECSIZE - 15*2 - sizeof(mon_errrec))                       \
        {                                                                       \
            register mon_putevent *reg_ptr = (mon_putevent *)mon_write_ptr;     \
            register short    *sen_fields = reg_ptr->fields;                    \
            monprintf("ReadSensor: mon_write_ptr = %d\n", mon_write_ptr);       \
            reg_ptr->cmd.type    = MONOP_PUTEVENT_INT;                          \
            reg_ptr->cmd.length  = sen_fields+3 - (short *)reg_ptr;             \
            mon_eventvector_count += reg_ptr->cmd.length*2;                     \
            reg_ptr->performer   = 0;                                           \
            reg_ptr->eventnumber = 8;                                           \
            reg_ptr->object       = htonl(((short)device<<16 )|                 \
                                  ((short)inumber  & 0xffff));                  \
            reg_ptr->initiator   = u.u_procp->p_pid;                            \
            reg_ptr->timestamp   = Timestamp;                                   \
            *(long *)sen_fields  = filepos;                                     \
```

```
            sen_fields          += 2;                                    \
            *sen_fields++         = actualcount;                         \
            if ( sen_fields > (short *)mon_eventvector_end )             \
                mon_write_ptr = mon_wraparound((unsigned char *)sen_fields);\
            else                                                         \
                mon_write_ptr = (unsigned char *)sen_fields;             \
            monprintf("    ReadSensor = %d\n", mon_write_ptr);           \
        }                                                                \
        else mon_oflow_count++;                                          \
        mon_semaphore--;                                                 \
    }                                                                    \
}          /* end readsensor */
#else
#define ReadSensor(a,b,c,d)
#endif
```

```
#ifdef MONITOR
#define NextComponent(device,inumber,filename)                                  \
if (*(mon_enablevector+0) & 1<<2)                                               \
{                                                                               \
    if (mon_semaphore++)                                                        \
    {                                                                           \
        if (mon_eventvector_count <                                             \
            MON_EVENTVECSIZE - 260 - sizeof(mon_errrec))                        \
        {                                                                       \
            register mon_putevent *reg_ptr  = (mon_putevent *)mon_write_ptr;\
            register short       *sen_fields = reg_ptr->fields;                 \
            register mon_string  sen_f_ptr = (mon_string)filename;              \
            register mon_string  sen_f_end = sen_f_ptr+127/sizeof(*mon_string);\
            register short sen_length;                                          \
            reg_ptr->eventnumber  = 2;                                          \
            reg_ptr->performer    = 0;                                          \
            reg_ptr->object       = htonl(((short)device<<16) |                 \
                                          ((short)inumber&0xffff));             \
            reg_ptr->initiator    = u.u_procp->p_pid;                           \
            reg_ptr->timestamp    = 0;                                          \
            do { *sen_fields++    = PackStr(sen_f_ptr); }                       \
            while ( NotEOS(sen_f_ptr, sen_f_end) );                             \
            *(sen_fields - 1)     &= ntohs(0xff00);                             \
            sen_length            = sen_fields - (short *)reg_ptr;              \
            reg_ptr->cmd.type     = MONOP_PUTEVENT_INT;                         \
            reg_ptr->cmd.length   = sen_length;                                 \
            mon_eventvector_count += sen_length*2;                              \
            if ( sen_fields > (short *)mon_eventvector_end )                    \
                mon_write_ptr = mon_wraparound((unsigned char *)sen_fields);\
            else                                                                \
                mon_write_ptr     = (unsigned char *)sen_fields;                \
        }                                                                       \
        else mon_oflow_count++;                                                 \
        mon_semaphore--;                                                        \
    }                                                                           \
}          /* end NextComponent */
#else
#define NextComponent(device,inumber,filename)
#endif
```

3.5. Blindprinter Output for the Example Sensors

```
command = external  length =     33     eventname = AcctHeader     performer =
9 53     object = 100 initiator =      0  timestamp = 1089898524     acct dat e
= 10487620162     kernel date = 10487620162 hostname = grant    init tex t =
Fri    Jun    14    14:03:05    1985          command    =    kernel    length    =
11     eventname    =    ReadSensor     performe  r    =         0    object    =
1280,2093  initiator    =       102  timestamp    =    1059    902835 filepos    =
198207actualcount =     24 command = external  length =     17     eventname =
UserXamplSensor       performe   r   =      953   object   =   101 initiator   =
0  timestamp    =     339019804         str_parm_1    =    acct    sh_parm_2    =
60      str_parm_3 = loop   lg_parm_4 = 101       command = kernel    length =
11     eventname = ReadSensor     performer =
 0     object  =  1344,187   initiator  =     952  timestamp  =  1059904085
filepos  =      301032actualcount  =    8192  command  =  kernel    length  =
11     eventname = ReadSensor     performer =
 0     object  =  1344,187   initiator  =     952  timestamp  =  1059955246
filepos  =      320110actualcount  =    8192  command  =  kernel    length  =
11     eventname = ReadSensor     performer =
 0     object  =  1344,187   initiator  =     952  timestamp  =  1059958996
filepos  =      302052actualcount  =    8192  command  =  kernel    length  =
11     eventname = ReadSensor     performer =
 0     object  =  1344,187   initiator  =     952  timestamp  =  1059959621
```

```
filepos = 320862actualcount = 8192 command = kernel length =
10 eventname = NextComponent performe r = 0 object =
1280,2 initiator = 952 timestamp = 0 filename = usr
```

Appendix B

Unix Manual Pages

These are the Unix manual pages for the parts of the system. It has a table of contents and a permuted index in the standard Unix format. The pages are in alphabetical order within the sections of the manual, except for the *intro* (1L) entry, which precedes the others in the section.

TABLE OF CONTENTS

PERMUTED INDEX

NAME

intro – introduction to the *Monitor* system tools

SYNOPSIS

toolname *options operands*

DESCRIPTION

The Monitor tools are used to examine the output from a run of *accountant*(1L). Each tool, with the exception of *tapehandle*(1L) and *enschema*(1L), reads a stream (see *stream*(5L)) from *stdin* and writes a stream to *stdout*. A stream consists of a schema (see *schema*(5L)) and a sequence of event records (see *sysl_monitor*(2L)). The event records are treated as *tuples* where the fields of an event record become *components* and the *relations* are determined by the *sensor* (see Generating Standard Sensors) that produced the event record. The schema is used to define the components for each relation to allow access by the tools.

The schema contained in the stream is modified whenever the components of a relation are changed and a new schema is produced for new relations in the output stream. The output record always contains the components corresponding to the struct **command** and the field **eventnumber** (see *sysl_monitor*(2L)), which serve to identify the relation. It is an error if the event record does not match the schema, or if there is no schema in the stream.

The options to a tool control the modes of operation of a tool. They may be specified anywhere on the command line. When an operand is a component name, it must be related to a relation. The general format is that the relation operand is specified first, followed by its components as separate operands.

COMMANDS

aggrop	apply one of (sum, average, count, min, max) to the stream
applyop	apply a given function to the stream
deschema	remove the schema from the stream into named file
enschema	prepend schema to stream
finitestate	apply a finite state machine to the stream
project	select or rearrange components
relretrieve	create a stream from an Ingres database
relstore	create an Ingres relation from a stream
select	select records from a stream based on a formula
streamprint	print records from a stream in a human readable format
streamconvert	convert records in a stream to host or net format
tapehandle	handle multiple tape archives of event records
union	conflate similar relations into a single relation

SEE ALSO

accountant(1L), syslocal(2L), sysl_monitor(2L), montools(3L), finitestate(5L), schema(5L), stream(5L), shutdownacct(8L)

Generating Standard Sensors, techreport 8. Stephen E. Duncan. University of North Carolina at Chapel Hill. July, 1985

EXAMPLE

The following session prints the average size of a read operation on a file basis:

```
tapehandle | enschema baseschema | aggrop -a ReadSensor object count
```

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

accountant – store event records in a file

SYNOPSIS

accountant [-a *accountantname*] [-k *kernelname*] [-t *initialtext*] [-x *tracelevel*]

DESCRIPTION

The accountant is a user-level process that interacts with sensors located in the Unix kernel and in other user-level processes. The primary task of the accountant is to spool event records generated by sensors onto a disk file. Additionally, the accountant also contains sensors that collect relevant information such as the current time, system load, etc.

The accountant first invokes a sensor which stores the current time, the compile time of the accountant (determined through the stat system call on the file named "accountant", as determined by *which)*, the compile time of the kernel (determined using the file name "/vmunix"), the name of the host (determined through gethostname), the current time in ASCII, and some initial text (defaulted to "Accountant started"). Several of these fields can be overridden using the -a, -k, and -t options.

The accountant then loops until terminated with the SIGTERM (15) signal, retrieving event records from the kernel and writing them to a set of temporary files of the form sendata.00000, where 00000 is modified to form a unique file name using *mktemp* .

The -x switch sets the level of debugging tracing: 0 for no tracing (the default), 10 for general control flow tracing, and 100 for detailed control flow.

RETURN VALUE

The accountant never returns; it must be signaled with SIGTERM (15).

SEE ALSO

which(1), gethostname(2), signal(2), stat(2), sysl_monitor(2L), mktemp(3), monlib(3L), shutdownacct(8L)

HISTORY

June, 1984 Stephen Duncan at the University of North Carolina, Chapel Hill
Added the signal handling and multiple file output,
Revised the accountant sensor.

May, 1983 Steve Rueman at the University of North Carolina, Chapel Hill
Added the spooling function.

June, 1981 Richard Snodgrass at Carnegie-Mellon University
Created.

NAME

acct – write event records to standard out

SYNOPSIS

acct [*numminutes*]

DESCRIPTION

This is a truncated version of accountant(1L). The accountant is a user-level process that interacts with sensors located in the Unix kernel. The primary task of the accountant is to spool event records generated by sensors onto a disk file. This particular version puts them to standard output. Additionally, the accountant also contains sensors that collect relevant information such as the current time, system load, etc.

The accountant first invokes a sensor which stores the current time, the compile time of the accountant (determined through the stat system call on the file named "acct", the compile time of the kernel (determined using the file name "/vmunix"), the name of the host (determined through gethostname), the current time in ASCII, and some initial text (defaulted to "Accountant started").

The accountant then loops for the number of minutes in its argument, or for the default of 20 minutes, retrieving event records from the kernel and writing them to standard out.

RETURN VALUE

The accountant returns 0.

SEE ALSO

accountant(1L), which(1), gethostname(2), signal(2), stat(2), sysl_monitor(2L), mktemp(3), monlib(3L), shutdownacct(8L)

HISTORY

Jan, 1985 Stephen Duncan at the University of North Carolina, Chapel Hill
Created.

NAME

aggrop – apply one of (sum, average, count, min, max) to the stream

SYNOPSIS

aggrop –sacnx *relation partition argument*

DESCRIPTION

Options

Specify which of the aggregate options to take.

–s	sum
–a	average value
–c	count of the number of occurances
–n	minimum value
–x	maximum value

All but **–c**, count, use the value of the component specified by *argument*. If no options are specified, **–c** is assumed.

Operands

Relation is the name of the relation on which the aggregate will be performed. If The *partition* specifies the component in *relation* on which to partition the records, e.g. initiator. The *argument* is the component in *relation* over which one or more of –sacnx is applied, and must be a numeric component.

Output

A table is printed to *stdout* whose first column is the *partition* value, and whose subsequent columns are, in order, the sum, average, count, minimum, and maximum. Only those columns whose option was specified appear in the output. The table can be further processed by other filters, if desired.

Error Output

A count of any error records detected in the stream is printed to *stderr*. If *relation* is specified as –, a count of non-conforming records, those which lack either the *partition* or the *argument* components, is also printed to *stderr*.

EXAMPLE

The following command produces a table of the average of the count component of the relation *ReadSensor* over the component initiator.

```
aggrop –a ReadSensor initiator count
```

This could produce the following table:

```
  0    25
315   512
 26    75
 13     8
  2    11
107  1245
```

SEE ALSO

intro(1L)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

applyop – apply a given function to the stream

SYNOPSIS

applyop [**–p** *arg*] *prog* [**–n** *name*] *relation component* [*component* ...] = *resultcomponent*:*type* [*resultcomponent*:*type* ...]

DISCRIPTION

Applyop extracts the components specified on the command line from the input stream (see *stream*(5L)), converts the values to character strings, and passes them to the function program on its *stdin*. Arguments specified with the **–p** option are passed unchanged to *prog* as command line arguments. The *stdout* of *prog* is interpreted as space separated fields comprising the *resultcomponents* which are appended to the relations of the output stream. The diagnostic output of the function is merged with the diagnostic output of *applyop*. This permits arbitrary functions to be applied to event records without writing specific stream handling routines for each function. It is assumed that the function knows what to expect as its arguments.

Options

–p Pass the next argument in the command line as a command line argument to the function. More than one **–p** may be specified. The arguments will be passed to the function in the order that they appear on the command line. All arguments passed to the function must be specified before the result components.

–n Instead of changing *relation*, use the next argument *name* as the name of a new relation consisting of tuples from *relation* with the result components appended.

Operands

arg	An optional argument to be passed unchanged to *prog*.
name	An optional name of a new relation to be created from the current tuple and the result component. If omitted, relation has the result components appended.
prog	The pathname of a program that performs the function.
relation	The name of the target relation, or – if for all relations.
component	A component in *relation* to be passed to *prog*.
=	A sentinal to signify the end of the list of components.
resultcomponent	A new component generated by *prog* of type *type* that is appended to *relation*.
type	The type of *resultcomponent*, which can be one of boolean, charstring, double (4 byte integer), int, and rational.

Output

Result components of the specified types appended to the specified relations. This includes modification of the associated schemas (see *stream*(5L) and *schema*(5L)).

EXAMPLE

The following applies an *awk* program to a stream via

```
applyop awk –c "–f awkfile" – initiator timestamp reltime:double
```

to append a component that contains the elapsed time since the last event of that initiator. Awkfile contains:

```
{
        if (save[$1] == 0)
                print 0
```

```
        else
                print ($2 - save[$1])
        save[$1] = $2
}
```

This program calculates the time between calls for a given *initiator*, and appends it to each event record as a new component reltime. The reltime for the first call is zero.

By piping the resulting stream through

aggrop -a - initiator reltime

(see *aggrop*(1L)) on the resulting stream, one can determine the mean time between system calls.

SEE ALSO

intro(1L)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

blindprint - print binary event records in human readable format

SYNOPSIS

blindprint [file ...]

DESCRIPTION

Blindprint reads each *file* of binary event records in sequence and displays it in human readable format on the standard output. If no file names are given, *blindprint* reads from the standard input. Thus

blindprint file

displays the file on the standard output, and

blindprint file1 file2 >file3

concatenates the first two files and places the result on the third.

Blindprint is designed to work the software monitoring system described in *SYSL_MONITOR* and *accountant*. The event records are produced by the monitoring system in binary format. *Blindprint* prints one event record per line with each field labeled.

SEE ALSO

accountant(1L), SYSL_MONITOR(2L), monlib(3L)

AUTHOR

3-June-84 S. Duncan, University of North Carolina, Chapel Hill
Created.

NAME

deschema – remove the schema from the stream into the named file *stdout*.

SYNOPSIS

deschema *schemafile*

DESCRIPTION

Operands

The pathname of the file to receive the schemas. Remove the schemas from the stream (see *stream*(5L)) into the named file, write the event record to *stdout*.

Output

The schema is written to *schemafile* and the eventrecords are written to *stdout*.

SEE ALSO

intro(1L)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

enschema – prepend the schema to the input event records

SYNOPSIS

enschema *schemafile* [*– file1 file2*]

DESCRIPTION

Operands

The pathname of the schema file (see *schema*(5L)), followed by optional file names. Input from *stdin* can be mixed with the file names by specifying – as a file name where *stdin* should appear. Specifying no file name defaults to *stdin*.

Output

A stream (see *stream*(5L)) is written to *stdout*.

SEE ALSO

intro(1L)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

finitestate – apply a finite state machine to a stream

SYNOPSIS

finitestate [**-k -n** *name*] *machinefile partition*

DISCRIPTION

Options

-k Keep tuples that would ordinarily be rejected. This permits conserving information that would otherwise be lost to the output stream.

-n Take the next argument as the name of the sentence header relation rather than FiniteState.

Operands

An optional name for the header relation, the pathname of a finite state machine file and a component on which to partition the event records. For each unique partition value, a separate instance of the finite state machine is executed.

Output

The schema is modified to hold the relation FiniteState which is created by this program if it does not already exist. Accepted sentences of event records, each preceded by a FiniteState event record, are written to *stdout*. FiniteState has the following fields (see *sysl_monitor*(2L):

cmd.type	– MONOP_PUTEVENT_EXT
cmd.length	– variable
eventnumber	– created unique in schema, if not already there
performer	– set to process id of program
object	– value of *partition*
initiator	– same as performer
timestamp	– taken from first event record in sentence
lasttimestamp	– taken from last event record in sentence
numberofevents	– length of sentence, in event records, excluding this one
partitionname	– *partition*, from the command line

Comments

This tool is needed for examining combinations of events. The other tools in general only deal with individual operations.

SEE ALSO

intro(1L)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

project – select or rearrange components.

SYNOPSIS

project [**–k –n** *name*] *relation component* [*component* ...] [– [**–n** *name*] *relation component* [*component* ...]]

DESCRIPTION

Options

–k Keep relations not listed in the operands. The default is to only write out the named relations.

–n Take the next argument as the new name for this relation. The default is to keep the old name. If issued with –k, Then the old relation is written out unchanged followed by the projected version under the new name.

Operands

Relations followed by the desired components. More than one relation can be specified by separating the relations with –. The components should appear in the desired order. Preceding each relation, an optional new name be be specified. The components for the struct command and the field **eventnumber** (see *sysl_monitor*(2L)) are always included and are always at the beginning of the record.

Output

A **stream** (see *stream*(5L)) of modified **event records** with appropriate **schema** (see *schema*(5L)). Each relation specified on the command line will contain only those components specified for it. Relations not specified on the command line will be discarded unless the **–k** option is specified, in which case they will be written out unaltered.

EXAMPLE

The following line projects the **timestamp** component of the *ReadSensor* and *WriteSensor* relations.

project ReadSensor timestamp – WriteSensor timestamp

Project can be especially useful when used before printing a **stream.**

SEE ALSO

intro(1L)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

relretrieve – create a stream of event records and schemas from the named Ingres database

SYNOPSIS

relretrieve *database* [*relation* ...]

DISCRIPTION

Operands

The database name and the relations to be put in the stream.

SEE ALSO

intro(1L)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

relstore – store the stream in the named Ingres database

SYNOPSIS

relstore *databasename*

DISCRIPTION

Operands

The name to use for the database.

SEE ALSO

intro(1L)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

select – select records from the stream based on a formula

SYNOPSIS

select [**–k –n** *name*] *relation formula*

DISCRIPTION

Options

–k Keep nonselected relations in the output. Only applicable if *relation* is not –.

–n Use the next argument as the name of the projected relation. If used with –k, the original relation is kept as well. Only applicable if *relation* is not –.

Operands

Name is used to replace *relation* as the name of the selected relation. This can't be used if *relation* is –. *Relation* specifies which relation contains the components in *formula*. If *relation* is specified as –, the formula is applied to all relations which have the requisite components. The *formula* consists of constants (numbers and quoted strings), component names (unquoted strings), regular expressions (delimitted by matched '/'s, see *regex*(3)), parentheses, comparison operators (! = > <), and logical operators (| &). It should be enclosed in single quotes to prevent the shell from expanding any characters. Since the *formula* is applied according to precedence, it is possible to specify components that only occur in some relations and still select other relations when the *formula* is used accross all relations via –.

Output

A stream of the selected event records.

EXAMPLES

To remove all the event records for a given initiator:

```
select – 'initiator != 27'
```

To select all *ReadSensor* events for the object 27514 by other than initiator 215:

```
select ReadSensor 'object = 27514 & initiator != 215'
```

To select all the tuples in the *NameStart* relation that have a filename that starts with "junk" and have an object between 1100 and 27514:

```
select Namestart 'filename = /junk*/ & object \!< 1100 & object \!> 27514'
```

SEE ALSO

intro(1L)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

streamconvert – convert the event records from or to network format

SYNOPSIS

streamconvert [–h –n]

DESCRIPTION

Since event records are binary data, their representation is machine dependent (see *byteorder*(3N)). *Streamconvert* uses the data in the schema (see *schema*(5L)) to convert each field in the event record to the appropriate format. The schema itself is always in human readable format, so that any machine can read it without conversion. Before a stream is written to tape, it should be converted to the network format via *streamconvert –n*. When a tape is read, it should, *enschema* should be run on it to creat a stream which should be run through *streamconvert –h* to put it into host format.

If transporting data between machines with the same format, conversion is not necessary.

Options

–h Convert the stream from network format to host format.

–n Convert from host to network format.

SEE ALSO

intro(1L), byteorder(3N)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

streamprint – print the event records in a stream in human readable format

SYNOPSIS

streamprint [–cklru]

DESCRIPTION

Read a stream (see *stream*(5L)) from *stdin* and print the event records to *stdout* in a human readable format. The event records are printed one per line, and the components are separated by tabs.

Options

Evaluation of a stream by a tool requires certain key fields so that relations can be properly identified. Therefore tools such as *project*(1L can't remove them from the relation. If the key fields are not really needed on the printed output, they will only clutter the output or force an additional filter to be used to remove them. Therefore the options [ckr] permit stripping off these fields when printing.

–k Omit the key fields (the command type, length, and event number) when printing. This is a superset of –c.

–c Omit the command type and length when printing. This is a subset of –k.

–l Begin each line with the relation name followed by two colons. Label the *components* of the event record using the *attributes* in the schema (see *schema*(5L)). This is the default mode. It will override a previous –u.

–r Omit the relation name when printing. This only is applicable when the output is labeled.

–u Don't label the *components*. This will override a previous –l.

For –l and –u, the options are evaluated in order, with the last option given taking precedence.

EXAMPLES

The following are samples of a single line of output from various combinations of options using the same event record as input.

streamprint

ReadSensor::cmdtype = 1 cmdlength = 11 eventnumber = 8 performer = 0 object = 851978 initiator = 12345timestamp = 49152046 filepos = 512 actualcount = 128

streamprint –u

1 11 8 0 851978 1234549152046 512 128

streamprint –r

cmdtype = 1 cmdlength = 11 eventnumber = 8 performer = 0 object = 851978 initiator = 12345timestamp = 49152046 filepos = 512 actualcount = 128

streamprint –c

ReadSensor::eventnumber = 8 performer = 0 object = 851978 initiator = 12345timestamp = 49152046 filepos = 512 actualcount = 128

streamprint –uc

8 0 851978 1234549152046 512 128

streamprint –k

ReadSensor::performer = 0 object = 851978 initiator = 12345timestamp = 49152046 filepos = 512 actualcount = 128

streamprint –uk

0 851978 1234549152046 512 128

SEE ALSO

intro(1L)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

tapehandle – handle multiple tape archives of event records

SYNOPSIS

tapehandle [**–s***size* **–n***number* **–b***blocksize* **–d***density* **–f***device*] **xc**

DESCRIPTION

This tool simplifies handling streams (see *stream*(5L)) that require more than one reel of tape.

OPTIONS

The size of the tape in inches, the number of tapes in the archive (extract only), the blocksize to use in bytes, the density of the tape, and the device to use. For create, use the options to determine how many bytes will fit on the tape. The defaults are:

size = 2400
blocksize = 12K
density = 1600bpi
device = /dev/rmt0

For extract, the defaults are:

device = /dev/rmt0

Operands

Specify **x** to extract a stream from the tapes or **c** to create a set of tapes from a stream.

Input specifications

The input is a sequence of event records which are read according to the data in the struct **command** (see *sysl_monitor*(2L)) and are in turn put out to tape. Calculate from the options or the defaults the number of bytes sufficient to fill a tape. When this number has been reached, prompt the operator for more tapes.

Output specifications

Read the event records from the tape according to the information in each record's command struct. When an EOT is read from the tape, prompt the operator to mount another tape, until the number of tapes specified has been reached, or the operator specifies there are no more tapes.

SEE ALSO

intro(1L)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

union – conflate similar relations into a single relation

SYNOPSIS

union *name relation1 relation2* [*relation3* ...]

DESCRIPTION

Union takes a stream (see *stream (5L)*) from *stdin*, creates a new relation *name* from *relation1*, *relation2*, ... , and writes the result to *stdout*. All relations in the list must have the same arity and component names.

Both the union relation and the original relations are present in the output.

Operands

Name is the name to call the resulting union, *relation1*, *relation2*, ... , are the names of the constituent relations.

EXAMPLE

Convert certain relations to the same arity and then combine them into a union. The target relations have the following components:

MySensor	*YourSensor*	*HisSensor*
cmdtype	cmdtype	cmdtype
cmdlength	cmdlength	cmdlength
eventnumber	eventnumber	eventnumber
object	object	object
initiator	initiator	initiator
performer	performer	performer
timestamp	timestamp	timestamp
name	value	weight
value	name	oldvalue
oldvalue	height	value
othervalue	oldvalue	name
history		otheruser

The tool *project*(1L) is used to change the relations to the same arity and to rearrange the components. *Union* then combines the relations into a single new relation.

```
project MySensor name value YourSensor name value HisSensor name value | \
union OurSensor MySensor YourSensor HisSensor
```

SEE ALSO

intro(1L)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

sysl_monitor – interact with kernel data collection

SYNOPSIS

```
#include <sys/syslocal.h>
#include <sys/mondefs.h>

result = syscall(SYSLOCAL, SYSL_MONITOR, buffer)
unsigned char *buffer;
int result;
```

DESCRIPTION

The function of this call is to facilitate communication between sensors embedded in executing programs and the monitoring process. The sensor stores data in the kernel of the operating system by invoking *SYSL_MONITOR* and the monitoring process, termed the *accountant*, retrieves it by also invoking *SYSL_MONITOR*. The accountant may also enable and disable sensors, and enable and disable all monitoring, through calls to *SYSL_MONITOR*.

The *buffer* is a pointer to a string of characters. The call returns an integer. *SYSL_MONITOR* is used to pass *event records* generated by sensors in the *target program* or in the kernel to the accountant. *Requests* for sensor enabling or disabling are passed from the accountant to the target program by first calling *SYSL_MONITOR* and signaling the target program. The target program then calls *SYSL_MONITOR* to retrieve the request. The target program can then modify its enable bits, located in the process's own address space.

A sensor in a target program sends information by calling *SYSL_MONITOR* with a pointer to an event record. Periodically the ACCOUNTANT calls *SYSL_MONITOR* to retrieve all the event records sent by sensors since the last time it did a read.

Two arrays are employed by *SYSL_MONITOR* to accomplish its function: one to hold requests, and one to store event records in the event buffer.

The buffer passed to *SYSL_MONITOR* is interpreted as a *command* beginning with the struct *mon_cmd* (see MONOP_INIT, below), which contains the *type* of command to *SYSL_MONITOR* and the *length* of the buffer, including the leading struct, in short integers. The length should be calculated by what is used in the buffer, rather than what is allocated.

Requests are used to enable and disable sensors in specific processes. The target process is identified by its *pid*, and the sensor to be enabled by its *eventnumber*. Requests are stored by passing a struct *request* to *SYSL_MONITOR* containing the target pid, the event number of the sensor, and a value which enables or disables the sensor. A process retrieves a request by calling *SYSL_MONITOR*, which searches the request buffer for the process's pid and returns the appropriate mon_request struct. This struct is then used by a routine in the process to enable or disable the indicated sensor. A process knows to call for a request by being signalled by the accountant, and the routine which traps this signal both executes the call and enables or disables the sensor.

```
struct mon_request {
        short targetpid;        /* process in which to change */
                                /* sensor (kernel = 0) */
        short eventnumber;
        short enablevalue;      /* = 1 to enable, = 0 to disable */
};
```

In the remainder of the section, the action taken for each one of the commands, (given in brackets) is discussed.

[MONOP_INIT] The command and event record buffers are initialized. If MONOP_INIT has already been called, a MON_ALRDY_INIT is returned; otherwise,

zero is returned. This call identifies the calling process as the accountant. All other commands return the error value MON_NOT_INIT if this command has not been executed.

```
struct mon_cmd {
        char type;              /* = MONOP_INIT */
        char length;            /* = (sizeof( struct mon_cmd ) + 1) / 2 */
};
```

[MONOP_PUTEVENT_INT]

A sensor wants to store an event record into the event buffer. The structure of an event record is as follows

```
struct mon_pevt {
        struct mon_cmd cmd;     /* type = MONOP_PUTEVENT_INT */
                                /* length determined by fields in sensor */
        short eventnumber;      /* id of sensor */
        short performer;        /* pid of performer of */
                                /* the operation */
        long object;            /* identifier of object */
                                /* operated on */
        short initiator;        /* pid of process requesting */
                                /* the operation */
        short fields[];         /* user-defined fields */
}
typedef struct mon_pevt mon_putevent;
```

SYSL_MONITOR sets the performer field in the event buffer to the PID of the calling process and the timestamp from the time in the kernel. If the event buffer is almost full, an error record is deposited in the event buffer instead of the event record, and MON_BUF_FULL is returned. Only one error record is deposited in the event record, indicating the loss of one or more event records. If there is enough room for the event record, a zero is returned unless MON_INIT had not previously been called.

```
struct mon_erec {
        struct mon_cmd cmd;     /* type = MONOP_OFLOW */
                                /* length = (sizeof(mon_errrec) +1)/2 */
        long val;       /* overflow count for MONOP_OFLOW */
}
typedef struct mon_erec mon_errrec;
```

[MONOP_GETEVENTS]

The ACCOUNTANT wants to retrieve all event records in the event buffer.

```
struct mon_gevt {
        struct mon_cmd cmd;     /* type = MONOP_GETEVENTS */
                                /* length = (sizeof(getevent) + 1) / 2 */
        short req_length; /* length of the remaining */
                                /* portion in short integers */
        short *acct_buf_ptr;    /* pointer to buffer to receive data */
```

```
};
typedef struct mon_gevt mon_getevent;
```

The event records received since the last GETEVENTS command are copied back into the buffer that *acct_buf_ptr* points to, and the number of short integers retrieved is returned. If the event records occupy more space than *req_length* short ints, then an integral number of event records is returned, occupying space not more than *req_length*. The buffer pointed to by *acct_buf_ptr* must have space enough for *req_length* short integers.

[MONOP_PUTREQ] A request is stored in the request buffer in the first available slot. The structure of a request is as follows:

```
struct mon_preq {
        struct mon_cmd cmd;             /* type = MONOP_PUTREQ */
                                        /* length = sizeof(putreq) */
        struct mon_request {
                short targetpid;        /* process in which to change */
                                        /* sensor (kernel = 0) */
                short eventnumber;      /* event to enable/disable */
                short enablevalue;      /* = 1 to enable, = 0 to disable */
        } req;
};
typedef struct mon_preq mon_putreq;
```

If the request buffer is full, a MON_REQ_OFLOW is returned to indicate the request was not stored; otherwise a zero is returned. The *targetpid* is the pid of the process whose sensors are to be affected. If it is zero, then it is the kernel's sensors that are affected immediately by the accountant. Otherwise the request is stored for retrieval by a MONOP_GETREQ.

[MONOP_GETREQ] The target program is calling to retrieve a request. The structure for a retrieval is identical to *mon_putreq* , above.

```
typedef struct mon_preq mon_getreq;
```

The request buffer is searched for a request with a matching process id *(pid)*. If the search is successful, the request is copied into the struct *req*, the corresponding entry in the request buffer entry is removed, and a zero is returned. If the request is not found, an error record is placed in the event buffer and a MON_REQ_NOT_FND is returned.

[MONOP_SHUTDOWN]

The accountant is calling to turn off monitoring. All further calls other than MONOP_INIT are ignored. In addition, all operating system sensors are turned off, and must be turned on again by the accountant. Any events left in the buffer from the last MONOP_GETEVENTS are lost, so all sensors should be disabled and MONOP_GETEVENTS used before MONOP_SHUTDOWN is used.

```
struct mon_cmd cmd;     /* type = MONOP_SHUTDOWN */
                        /* length = (sizeof(struct mon_cmd) + 1) / 2 */
```

RETURN VALUE

The call returns a negative integer on error, the values of which may be found in <sys/mondefs.h>. If it succeeds it returns a non-negative integer which is command specific (see above). Some calls have the side effect of changing the buffer passed in the call.

ERRORS

If a process other than the accountant issues the GETEVENTS, PUTREQ, or SHUTDOWN commands, a MON_NOT_ACCNT is returned and the command is not executed. If a command other than one of the commands listed above is given, then MON_INV_CMD is returned. The error value MON_SYS_ERR means that a system call error was found and *perror* (see intro(2)) can be used to print the system error message. If the error value MON_CONCURRENCY_ERR is returned, then a problem in the kernel sensors has been detected. The kernel sensors will have been disabled and no more commands may be given to *SYSLMONITOR* . The system must be rebooted to restore the monitor. The remaining errors are command-specific (see above).

SEE ALSO

accountant(1L), blindprint(1L), syscall(2), syslocal(2L), monlib(3L)

HISTORY

15-May-83 Created. R. Snodgrass, D. Doerner, R. Fisher, S. Reuman, University of North Carolina, Chapel Hill

MODIFIED

25-Jan-85 Changed event record buffer to a ring buffer, modified sensor enabling code, changed to use structures, added security.
S. Duncan, University of North Carolina, Chapel Hill

14-Jun-85 Fixed various bugs, modified to work on Suns as well as vaxes.
S. Duncan, University of North Carolina, Chapel Hill

NAME

syslocal – indirect local system call

SYNOPSIS

#include <sys/syslocal.h>
syscall(SYSLOCAL, number, arg, ...)

DESCRIPTION

syslocal performs the local system call whose interface has the specified *number*, and further arguments *arg*.

The r0 value of the system call is returned.

DIAGNOSTICS

When the C-bit is set, *SYSLOCAL* returns –1 and sets the external variable *errno* (see *intro*(2)).

If the specified code is not a valid *SYSLOCAL* system call, *SYSLOCAL* returns error code EINVAL; see *intro(2)*.

HISTORY

22-Jun-84 Tim Seaver (tas) at University of North Carolina
Created.

NAME

str_open, str_fopen, str_schemaread, str_schemawrite, str_read, str_write, getrelationbysensorid, geterelationbyname, getrelation, rmrelationbyname, setposition, getdomainbyname, rmdomainbyname, copydomain, copyrelation, copyschema, readrecord, writerecord – Monitor system stream and tuple operations

SYNOPSIS

```
#include <monitor/montypes.h>
#include schema_idl.h
#include streamio.h

Mstream * str_open(fp)
FILE *fp;

Mstream * str_fopen(filename,mode)
char *filename, *mode;

database str_schemaread(sp)
Mstream *sp;

int str_schemawrite(sp)
Mstream *sp;

int str_read(sp,tp)
Mstream *sp;
tuple *tp;

int str_write(sp,tp)
Mstream *sp;
tuple *tp;


#include <monitor/montypes.h>
#include schema_idl.h
#include tuple.h

relation getrelationbysensorid(schema, sensorid)
database schema;
int sensorid;

relation getrelation(record,schema)
short *record;
database schema;

relation getrelationbyname(schema,name)
database schema;
char *name;

void rmrelationbyname(schema, name)
database schema;
char *name;

void setposition(tp)
tuple *tp;

attribute getdomainbyname(rp,domname)
relation rp;
char *domname;

void rmdomainbyname(rp,dname)
relation rp;
char *dname;
```

```
attribute copydomain(ap)
attribute ap;

relation copyrelation(rp)
relation rp;

database copyschema(schema)
database schema;

tupleprint(fp,tp,label)
FILE *fp;
tuple *tp;
int label;


#include <monitor/montypes.h>

int readrecord(fp,recd)
FILE    *fp;
mon_putevent *recd;

int writerecord(fp,recd)
FILE *fp;
mon_putevent *recd;
```

DESCRIPTION

To use the procedures, the *include* files for each set of procedures must be specified, and the library must be linked. The linking is done by specifiying "–lmontools" to the C compiler if the library is installed, or "*LIBDIR*/libmontools.a" if it isn't installed, where *LIBDIR* is the path of the directory where the library exists.

Stream Operations

str_open returns a stream (see *stream*(5L)) associated with the open file pointer *fp*. NULL is returned if *fp* is NULL.

str_fopen returns a stream for the file *filename* opened in mode *mode* (see *fopen*(3S)). If an error is detected when opening *filename*, the global value of **errno** (see *intro*(3)) is set and NULL is returned.

str_schemaread returns the schema (see *schema*(5L)) read from the front of stream *sp*. It will only read the schema once. If an attempt is made to read the schema more than once, an error message is printed to *stderr* and NULL is returned. NULL is also returned is the schema can't be read.

str_schemawrite writes the schema associated with stream *sp* on the front of *sp*. The schema will only be written once. A zero will be returned on successfully writing the schema. Attempts to write the schema more than once results in an error message printed to *stderr* and a return value of –1.

str_read reads the tuple (see *intro*(1L)) *tp* from the stream *sp*. If the schema has not already been read from *sp*, then it will be read first. *str_read* returns the number of chars read. Zero represents *EOF*. If there is no schema for the stream, STRIO_ESCHEMA is returned. If an error is detected while reading *tp*, STRIO_EREAD is returned. *str_read* exits with a status of 1 on a buffer overflow.

str_write writes the tuple *tp* on the stream *sp*. If the schema has not already been written to *sp*, then it is written first. If there is no schema for *sp*, STRIO_ESCHEMA is returned.

Tuple Operations

getrelationbysensorid returns the relation in *schema* with a sensor id of *sensorid*. NULL is returned if no match is found.

getrelation returns the relation in *schema* using the data in *record*. NULL is returned if no match is found.

geterelationbyname returns the relation in *schema* with the rel_name of *name*.

rmrelationbyname removes the relation with rel_name of *name* from *schema*. If the named relation isn't in *schema*, there is no effect.

setposition updates the positions of the domains in the tuple.

getdomainbyname returns the attribute in the relation *rp* for the domain named *domname*. NULL is returned if no match is made.

rmdomainbyname removes the attribute for the domain *dname* from the relation *rp*. There is no effect if the domain isn't in the relation.

copydomain returns a copy of the attribute for the domain *ap*. Only the attr_name is shared in memory. This is needed when a new relation is built that is similar but not the same as an old relation.

copyrelation returns a copy of the relation *rp* . Only the rel_name is shared in memory. This is needed when a relation needs to be modified in one schema but not in another.

copyschema returns a copy of the database(schema, see (*schema*(5L))) *schema*. Only the database_name is shared in memory. This is needed if the schema is to be modified between input and output.

tupleprint prints the event record from tp in a human readable format to file fp. If label is set to PRINTLABELS, then each field in the record and the record itself is labelled. A −1 is returned if the file doesn't exist or if there is an error in tp.

Event Record Operations

readrecord writes the mon_putevent (see *sysl_monitor*(2L)) *recd* to the FILE *fp*. It returns the number of chars read or zero at *EOF*. A negative return value signifies an error.

writerecord writes the mon_putevent *recd* to the FILE *fp*. It returns the number of chars written or a negative value if an error is detected.

SEE ALSO

accountant(1L), blindprint(1L), intro(1L), syslocal(2L), sysl_monitor(2L), finitestate(5L), schema(5L), stream(5L), shutdownacct(8L)
Generating Standard Sensors, techreport 8. Stephen E. Duncan. University of North Carolina at Chapel Hill. July, 1985

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

finitestate – finite state machine description format

DISCRIPTION

Finitestate(1L) reads a finite state machine description and executes the machine against a **stream** (see *stream*(5L)).

Each line of the description is a transition consisting of a label for the state, a label for the next state, and a list of triples (*relation,domain,value*) which specify whether to take the transition.

state nextstate [*relation domainname value ...*]

state and *nextstate* are integers, *relationname* and *domainname* are character strings, and *value* has the same type as the domain. If the triples are completely omitted, then the transition is always taken. A member of the triple to be ignored can be indicated by –. The following are the possible combinations:

relation – –	True if the current event record is in *relation*.
– *domain value*	True if the current event record has *domain* in it and *domain* has value *value*.
– *domain* –	True if the current event record has *domain* in it.

If *value* is a backslash followed by a relation operator, the *domain* value is compared to the *domian* value of the previous tuple in the sentence, rather than with *value* itself. The operator must be one of >, <, or = in combination with ! (logical not), This provides for determining equivalence classes.

The first transition in the file that has all of its conditions met is the one that will be taken. The states must be specified in ascending order.

The starting state is indicated with a *state* of "1", accepting states are indicated with a *nextstate* of "0", and rejecting states are indicated with a negative *nextstate*, or by the lack of a transition line for the input. The internal format of the finite state machine does not allow for non–deterministic machines.

EXAMPLE

The following finite state machine accepts sentences from a stream where all the tuples have the same value in the initiator domain. Each accepted sentence represents the sequence of events used by the kernel follows a pathname to open a file, and is a measure of locality of reference for file names. An accepted sentence begins with *NameStart*, followed by a sequence of *NextComponents*, and ends with *OpenSuccessful*. It is possible to have another *NameStart* in the sentence if the path contained soft links (see *ln*(1)) and it is possible to have *ReadSensor* and *WriteSensor* when additional blocks from the file system are needed to trace the path. If any other relation is detected in mid-sentence, the sentence is discarded.

```
1 2 NameStart – –
1 1
2 2 NextComponent – –
2 0 OpenSuccessful – –
2 3 ReadSensor – – WriteSensor – –
2 1
3 2 Namestart – – NextComponent – –
3 3 ReadSensor – – WriteSensor – –
3 1
```

If the resulting output is piped into *aggrop*(1L), the length of the sentence will be known.

SEE ALSO

finitestate(1L)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

schema – IDL description of event records

DESCRIPTION

The **schema** is generated automatically by the sensor compiler. It contains the names and attributes of each field for the event record produced by each sensor. This allows routines to understand event records without having to hard code the attributes. The **schema** is in external *IDL* format. The **schema** is read and written through the *IDL ports* **inputandoutput**.

FILES

schema_idl	The description of the schema in *IDL*
schema_idl.h	Generated header file
schema_idl.o	Support routines

SEE ALSO

idl(1), intro(1L), snlproc(1)

A Tutorial Introduction to Using IDL, techreport 1. William B. Warren, Jerry Kickenson, and Richard Snodgrass. University of North Carolina at Chapel Hill. November, 1985

Using IDL with C (Version 1.0), techreport 6. Tim Maroney and Karen Shannon University of North Carolina at Chapel Hill. June, 1985

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

stream – the data structure used by the SoftLab Monitor system

DESCRIPTION

A stream consists of a schema (see *schema*(5L)) terminated by a marker and followed by a sequence of event records (see *sysl_monitor*(2L)). The event records are treated as tuples whose domains are the fields of the event records and whose relations defined by which sensor created the record. The Monitor tools read in the stream and use it to interpret the following event records. Any domains added or removed from a relation must be reflected in the schema. A set of library routines to manipulate streams is found in *libmontools*(3L).

SEE ALSO

intro(1L), sysl_monitor(2L), libmontools(3L), schema(5L)

AUTHOR

Stephen E. Duncan, University of North Carolina at Chapel Hill

NAME

shutdownacct – emergency close down of the monitoring system

SYNOPSIS

/etc/shutdownacct

DESCRIPTION

Shutdownacct provides an emergency shutdown procedure which a super-user can use to stop the monitoring system in the event that something has happened to the *accountant* (see accountant(1L)).

SEE ALSO

accountant(1L)

AUTHOR

14-June-85 S. Duncan, University of North Carolina, Chapel Hill

Appendix C

Installing the Monitor System, Release 1.3

This is SoftLab internal document 14, distributed with the system to explain installation and operation of the system.

Abstract

Directions for installing the Monitor system from a distribution tape. Includes directions on the insertion of sensors into kernel source code, compiling the new system, and how to operate the accounting process.

Changes from release 1.2

- Errors in the event vector handling have been corrected
- The sensor FileClose is now placed after the NULL test
- The granularity of calls to SYSL_MONITOR is reduced to 3 seconds
- The source code for local_syscalls.c has been cleaned up
- Tests for concurrancy have been added to the system.

Installation

The *Monitor* system is a collection of programs and routines for monitoring operating systems and user processes. It is based around a *monitor* system call that controls access to the system's data buffers and operations and a user program, called the *accountant*, that controls the system call. *Sensors*, in the form of macros, are inserted into the target routines, which are enabled and disabled by the monitor at the direction of the accountant. The accountant periodically writes the monitor's buffers out in a raw format. The program `blindprint` prints the data in human readable format. The routines in `monlib` allow manipulation of the raw data by other programs.

The system is designed to work under Unix 4.2 on Vaxes and under Unix 4.2, Sun release 1.4 on SUn workstations. The distribution tape holds a single directory, `tempmon`, which holds all of the source for the Monitor system. The system consists of the following parts:

The accountant – `acct.c acct_sensors.h`
The sensor macros – `kern_sensors.h ufs_sensors.h sys_sensors.h`
The system calls SYSLOCAL and SYSL_MONITOR – `local_syscalls.c`
The kernel files to be modified –
`kern_descrip.c sys_generic.c ufs_nami.c ufs_syscalls.c ufs_nami.c init_sysent.c`
A set of library routines – `readrecord.c printevents.c dumprecord.c`, found in `monlib.a`
A non-interpretive printer – `blindprint.c`
Various include files – `syscalls.h mondefs.h monerrcds.h monops.h montypes.h`
A set of manual pages and the documents "Generating Standard Sensors" and "Installing the Monitor system". Source for the manual pages is provided.
Files for modifying the kernel configuration –
`file.MONITOR shortalign.c` with a `Makefile`
Files for installing the kernel patches – `patch.c`

These are organized within `tempmon` into the following directories:

`monitor`	mon prefixed include files
`monsys`	C files and modified kernel files
`monsys/doc`	manual pages
`Patch`	Larry Wall's patching program
`h`	kernel sensors and `syscalls.h`

The distribution tape contain 186Kbytes of data. The installed code occupies about 400K plus the size of the directory to hold the new kernel (about 1200K). The data produced by the system can be enormous: over 10K per second of operation on a loaded system.

To install the Monitor system, perform the following steps:

(1) Change your current working directory to a directory where you want the contents of the distribution tape to be stored until the installation is complete. The contents will be stored in a directory named `tempmon` in the current working directory.

```
cd somedir
```

(2) Mount the tape, and ensure that it is not write–enabled. The tape is read by typing

```
tar x
```

(3) Change to the new directory

```
cd tempmon
```

(4) The monitor system has a set of dependencies that must be investigated before continuing. In unmodified versions of 4.2BSD, syscalls.c and init_sysent.c each have a table of 150 entries. This installation adds entry 151. If you installation has already used entry 151, then you will have to change the following to reflect a different entry number. The new entry number should be one greater than the highest allocated entry number. Note that additional local system calls can be added with changes only to `syslocal.h`.

```
tempmon/monsys/syslocal.h
tempmon/monsys/syscalls.c.patch
tempmon/monsys/init_sysent.c.patch
```

(5) The directory `tempmon` contains a makefile with default directories and flags for storing the system. Check in your system directory, usually `/sys`, to see if there are any directory conflicts, and in the makefile for the current kernel image for any define conflicts. This is very important because any directories with conflicting names may be damaged. Edit `tempmon/makefile` accordingly. In general, `KERNEL` should be set to the standard system name.

```
KERNEL = GENERIC          #Name of target kernel, used as basis for monitor kernel
MONITOR = MONITOR         #Name for Monitor kernel
MONDEF = MONITOR          #Define for kernel's makefile
MONINCLUDE = monitor      #Name of directory for monitor include files
MONSYS = monsys           #Name of directory for Monitor system
SYS = /sys                #Name of system directory
```

(6) The patches to the system files are applied to the following files, if you wish to add or remove files, edit `makefile`. The files `syscalls.c` and `init_sysent.c` are required to support the system call.

```
PATCH = kern_descrip.c ufs_syscalls.c ufs_alloc.c ufs_nami.c \
        sys_generic.c syscalls.c
```

The original files will not be affected.

Superuser privileges are required to perform the remainder of the installation.

(7) The first task is to check out the properties of the system:

```
make config
```

(8) Move the directories and files into position and compile the user files:

```
make install
```

This also ensures that the new defines are in place. In general, if you make a minimum of changes to the defaults, you will see a large number of error messages from *make*. This is because a number of tests are made to determine the files status and is normal. *Make* is directed to ignore these errors. Others will cause it to abend. You should save the output from the make to insure that all went well.

(9) Create the monitored kernel by typing:

```
make new
```

This applies patches to the files, configures and makes a new kernel file. Check the files `depend.out` and `make.out` in `$(SYS)/$(MONITOR)` to see if all went well. The target files should show up in `$(SYS)/$(MONITOR)/makefile` with different dependencies than in the regular kernel. The end of make.out should have *loading vmunix* at the end of it.

(10) The directory `tempmon` is no longer needed and can be removed.

```
rm -rf tempmon
```

Before repeating any steps involving *make*, certain actions must be taken to insure proper installation.

(1) The *make config* step can simply be rerun.

(2) The *make install* step requires the the files with the extension `.bak` replace their equivalents that lack `.bak` in the three directories `tempmon/h`, `tempmon/monitor`, and `tempmon/monsys`. The directories `$(SYS)/$(MONSYS)` and `$(SYS)/($MONINCLUDE)` and the link `/usr/include/$(MONINCLUDE)` should be deleted.

(3) The *make config* step requires that the files in `$(PATCH)` (see above) should be removed from `$(SYS)/$(MONSYS)` and the files `$(MONITOR)` and `files.$(MONITOR)` be removed from `$(SYS)/conf`.

The whole procedure should only take about an hour on an unloaded Sun, most of which is for compiling `vmunix`. Vaxes will take about two hours, since they generally have more source files to be compiled. While this creates a bootable image, it doesn't bring it up. Before continuing, you should read the manual pages for *shutdown*, and *halt*, section 6 in the article "Installing and Operating 4.2BSD on the VAX", and browse through the article "Building 4.2BSD UNIX Systems with Config".

The following steps bring up the monitor system.

(1) Move the new kernel to the root file system.

```
mv $(SYS)/monitor/vmunix /monvmunix
```

It is important to not overwrite the existing kernel image. Save it just in case.

```
ln /vmunix /regvmunix
```

(2) Bring the system down.

```
shutdown -h +15 Putting in new kernel
```

If no one is around, you can say *now* instead of *+15*.

(3) You will have to find out what type of disk your root file system is on. The Unix command *df* will show the device names for the file systems. The root file system is on the first *0a* partition listed. Replace the two letter prefix for *dk* in the commands below. Halt the machine. The commands vary from machine to machine, so check with your systems programmer before trying this. The machine's prompts are in bold. Commands will be given first for Vaxes and then for Suns.

```
>>> P          These two lines stop the CPU
>>> H
```

It then prints messages that the CPU is halted.

```
>>> B ANY          for a VAX780 or 730
      or
>>> B/3            for a VAX750
Boot: dk(0,0)monvmunix -s
```

For Sun workstations, the operation is simpler. The machine is halted by holding down the *L1* function key and hitting *A*. From there:

```
>B dk(0,0,0)monvmunix -s
```

From here on Suns and Vaxes behave the same. The system will come up now in single user mode. Examine the various file systems and use different commands to see if everything is all right. Avoid commands such as *w*, *who*, *vmstat*, and *ps* which require that the kernel be named `vmunix`.

(4) If everything looks alright, move in the new kernel.

```
rm /vmunix          Remember a copy is saved in /regvmunix
ln /monvmunix /vmunix
```

Since you saved the old *vmunix*, you can switch back and forth between the two versions fairly easily.

(5) Reboot the system again. This is much easier.

```
reboot
```

When the system comes up, it will now be the MONITOR system.

The accountant resides in $(SYS)/monsys, along with other support files. This is not a full implementation of the accountant, but does handle the basic tasks of managing the monitor system call. The basic method of operation is to run the accountant for a given period of time, collecting the raw data from standard out. The raw data may then be examined using `blindprint` or by programs using the routines in `monlib`.

```
acct 120 > acct.rdata &  Collect data for 120 minutes
Wait for it to finish
blindprint acct.rdata | lpr     Print the data
```

A great quantity of data can be produced by an active system, so you should try smaller time periods at first. The accountant can be stopped with signals, but data will keep being fed into the monitor if it doesn't terminate normally. The program `shutdownacct` can be used to close down the monitor. Check the manual page for *acct* for details of its operation.

The distribution contains the following kernel sensors which are installed in the Unix file system:

ReadSensor
WriteSensor
FileClose
INodeCreate
INodeDelete
OpenSuccessful
NameStart
NextComponent

All the sensors, except for OpenSucessful, use the device/inode numbers to uniquely identify the file. OpenSuccessful, which is placed in the file `ufs_syscalls.c`, uses the associated series of event records produced by NameStart and NextComponent to identify the file. NameStart is placed where the code begins to look up a path name, while NextComponent records processing of each part of the path name. Both are in the file `ufs_nami.c`. INodeCreate and INodeDelete are placed in the file `ufs_alloc.c` and record the actual creation and deletion of files. Deletion only occurs when the last reference to a file is removed. ReadSensor and WriteSensor are placed in the file `sys_generic.c` and are activated for all reads and writes on file descriptors. FileClose is place in `kern_descrip.c` and is activated when a file descripter is closed.

Appendix D

Source Code Listings

This is a listing of a selection of the code for the project. The code is printed by files within grouped by directories. A directory may contain the files for a specific command or just a set of related files. Each directory starts out on a new page with its name in bold face. Files within a directory have their names set off between two horizontal lines. The directory name for a command or library is the command or library name, while the other directory names show part of their path, e.g. *sys/monsys*. Not include in this listing are patches to the kernel routines, since this code might be proprietary.

Table of Contents

ACCOUNTANT

README

```
This directory contains the accountant with its necessary sensors.
Basic organization is to divide into modules such that only some
have to be looked at when studying system.
Each decomposed module gets the global include files, if necessary,
and whatever external variables are necessary.
        CMU.c                   - controls CMU operation
        CMUconst.h              - CMU specific constants
        CMUvars.h               - CMU specific variables
        DoUnixProto.c           - Unix event record handling
        Finish.c                - Unix termination routine
        ProcessCmd.c            - Simon interface
        RCS/
        README
        SendDataRecords.c       - Simon interface
        SendError.c             - Simon interface
        SendEventRecord.c       - Simon interface
        SensorControl.c         - Enables and disables kernel sensors
        WriteEventRecord.c      - Unix output handler
        accountant.h            - Sensors for the accountant
        accountant.sen          - Definition of accountant sensors
        const.h                 - global constants
        enet.h                  - ethernet stuff
        ipc.h                   - CMU ipc stuff (4.1?)
        main.c                  - main routines, was accountant.c
        makefile
        old/                    - obsolete stuff
        queue.h                 - CMU include file
        vars.h                  - global variables

These files make up a fake kernel that can be used to test the accountant.
        minikern.c              - simulates some kernel calls
        sys.c                   - holds sensors
        ufs.c                   - holds sensors
        kern.c                  - holds sensors
        local_syscalls.c        - bring over from /sys/monsys
```

makefile

```
# To compile the UNC version, set WHERE to -DUNC -DMONITOR
#       and then make accountant.

CMUHEADERS = const.h CMUvars.h CMU.c
SIMON = SendDataRecords.c SendError.c SendEventRecord.c ProcessCmd.c
OBJSIMON = SendDataRecords.o SendError.o SendEventRecord.o ProcessCmd.o
STANDALONE = DoUnixProto.c Finish.c WriteEventRecord.c
OBJSTANDALONE = DoUnixProto.o Finish.o WriteEventRecord.o SensorControl.o
OBJMINI = minikern.o ufs.o sys.o kern.o local_syscalls.o
MAXFILESIZE =

CFLAGS = -g
# Make this line blank for compiling at CMU
WHERE = -DUNC -DMONITOR



# Change this to default module
accountant:     uncstandalone

# For the UNC Unix standalone system
uncstandalone:  main.c $(OBJSTANDALONE)
        cc -o accountant $(WHERE) -DSTANDALONE main.c $(OBJSTANDALONE)

# For the UNC Unix monitor driven system
uncsimon:       main.c $(OBJSIMON)
        cc -o accountant $(WHERE) -DSIMON main.c $(OBJSIMON)

# For the CMU system
cmu:            main.c $(OBJSIMON)
        cc -o accountant main.c $(OBJSIMON)



# These are the Unix modules

DoUnixProto.o:  DoUnixProto.c
        cc -c DoUnixProto.c $(CFLAGS) -DSHORTALIGN -DSTANDALONE $(WHERE)
Finish.o:       Finish.c
        cc -c Finish.c $(CFLAGS) -DSHORTALIGN -DSTANDALONE $(WHERE)
WriteEventRecord.o:     WriteEventRecord.c
        cc -c WriteEventRecord.c $(CFLAGS) $(MAXFILESIZE) -DSHORTALIGN -DSTAND
ALONE $(WHERE)

SensorControl.o:        SensorControl.c
        cc -c $(CFLAGS) $(WHERE) -DSHORTALIGN -DSTANDALONE SensorControl.c
```

```
# These are for use with the Simon monitor

SendDataRecords.o:      SendDataRecords.c
        cc -c SendDataRecords.c $(WHERE)

SendError.o:    SendError.c
        cc -c SendError.c $(WHERE)

SendEventRecord.o:      SendEventRecord.c
        cc -c SendEventRecord.c $(WHERE)
ProcessCmd.o:           ProcessCmd.c
        cc -c ProcessCmd.c $(WHERE)



# These files form the minikernel, for testing without installation

minikern.o:     minikern.c
        cc -c -g -DMONITOR -Usun -DKERNEL -DSHORTALIGN minikern.c

ufs.o:          ufs.c
        cc -c -g -DMONITOR -Usun -DKERNEL -DSHORTALIGN ufs.c

sys.o:          sys.c
        cc -c -g -DMONITOR -Usun -DKERNEL -DSHORTALIGN sys.c

kern.o:         kern.c
        cc -c -g -DMONITOR -Usun -DKERNEL -DSHORTALIGN kern.c

local_syscalls.o:       local_syscalls.c
        cc -c -g -DMONITOR -DKERNEL -Usun -DSHORTALIGN local_syscalls.c

testaccountant: main.c $(OBJSTANDALONE) $(OBJMINI)
        cc -g -o testaccountant $(WHERE) -Usun -DSTANDALONE \
                -DSHORTALIGN main.c $(OBJSTANDALONE) $(OBJMINI)



# These are the include file dependencies, but are not clever
# about ifdef's and the like.

DoUnixProto.o:  /usr/include/monitor/monops.h
DoUnixProto.o:  /usr/include/monitor/montypes.h
DoUnixProto.o:  /usr/include/sys/syslocal.h
DoUnixProto.o:  const.h
Finish.o:       /usr/include/monitor/monops.h
Finish.o:       /usr/include/monitor/montypes.h
Finish.o:       /usr/include/sys/syslocal.h
Finish.o:       const.h
```



```
WriteEventRecord.o:     const.h
main.o:         /usr/include/monitor/monops.h
main.o:         /usr/include/monitor/monerrcds.h
main.o:         accountant.h
main.o:         acc_set.h
main.o:         const.h
main.o:         queue.h
main.o:         enet.h
main.o:         CMUconst.h CMUvars.h CMU.c
SendDataRecord.o: const.h
SendDataRecord.o: vars.h
SendError.o:    const.h
SendError.o:    vars.h
SendEventRecord.o: const.h
SendEventRecord.o: vars.h
ProcessCmd.o:   ipc.h
ProcessCmd.o:   enet.h
ProcessCmd.o:   queue.h
ProcessCmd.o:   CMUconst.h
ProcessCmd.o:   CMUvars.h
ProcessCmd.o:   const.h
ProcessCmd.o:   vars.h
ProcessCmd.o:   /usr/include/monitor/monops.h
ProcessCmd.o:   /usr/include/monitor/monerrcds.h
```

---

const.h

```
/*****************************************************************************/
/*                              const.h                                      */
/* This file contains the definitions of constants used by the accountant.   */
/*                                                                           */
/* Original author: Richard Snodgrass                                        */
/*****************************************************************************/

/************************* Command Types *********************************/

/* Sent from Simon to the resident monitor      additional argument(s)     */
#define EndCommand              0       /* none                             */
#define AdjustObjectCommand     1       /*object-id,event-id,internal value*/
#define CheckPointCommand       2       /* object-type                     */
#define ReadEntryCommand        3       /* entry-id                        */
#define WriteEntryCommand       4       /* entry-id value                  */
#define SampleCommand           5       /* object-id entry-id              */
```

```
/* Sent from Simon to Simon Accountant     additional argument(s)          */
#define TerminateAccCommand      128     /* none                           */
#define SetTraceCommand          129     /* new trace value                */
#define AddObjectToClass         130     /* object, class, internal flag   */
#define AddEventToClass          131     /* event, class, internal flag,
                                            timestamp flag                 */
#define AddDomainToEvent         132     /* domain, event                  */
#define StartNormalProcessing    133     /* none                           */
#define InitAccountant           134     /* InitCount, InitTime            */

/* Sent from Simon Accountant to Simon     additional argument(s)          */
#define InitDataWords            3
#define InitDataRecord           128     /* InitDataWords worth of integers */
#define AccAbortDataRecord       129     /* sequence number                */

/********************** Data Record Types ***********************************/

/* Sent from the resident monitor to Simon  additional argument(s)         */
#define LastRecord               0       /* word-count                     */
#define EventRecord              1       /* many parameters                */
#define ReportDataRecord         2       /* entry-id value                 */
#define CheckDataRecord          3       /* timestamp                      */

#define ErrorDataRecord          5       /* error-num additional-info      */
#define NewNameDataRecord        6       /* object-id timestamp            */

/***************************** Error Codes **********************************/

/* Sent from Simon Accountant to Simon     additional argument(s)          */
#define BadClass                 128     /* class number                   */
#define BadDomain                129     /* domain index                   */
#define BadEventNumber           130     /* event number                   */
#define BadDomainType            131     /* domain, type                   */
#define TooManyObjectsInClass    132     /* class                          */
#define UnIdentifiedObjectInEvent 133    /* event                          */
#define UnIdentifiedEvent        134     /* class, event                   */
#define TooManyEventsInClass     135     /* class                          */
#define EventHasBadDomains       136     /* event                          */
#define BadClassType             137     /* classtype                      */
#define UnIdentifiedObject       138     /* object                         */
#define BadInfoArrayIndex        139     /* index                          */

/***************************** Exit Codes ***********************************/

#define StdTermination           0
#define ResidentMonitorEnd       -1
#define CantLocateAccData        -2
#define CantOpen                 -3
#define CantWrite                -4
#define BadClose                 -5
```



```
/************************ Domain Types **********************************/

#define IntDomain                0
#define DblIntDomain             1
#define StringDomain             2
#define MaxDomainType            2

/************************ Array Sizes ***********************************/

#define MaxClassTypes            2       /* ClassArray[x][]                */
#define MaxNumClasses            10      /* ClassArray[][x]                */
#define NumEventsPerClass        20      /* ClassArray[][].EventID[x]      */
#define NumObjectsPerClass       100     /* ClassArray[][].SName[x]        */
#define NumEventsPerObject       128     /* ClassArray[][].position[x]     */
#define MaxNumEvents             255     /* EventArray[x]                  */
#define MaxNumDomains            10      /* EventArray[].Domains[x]        */
#define MaxInfoItems             5       /* InfoArray[x]                   */
#define MaxEventRecordLength     50000   /* EventRecBuf[x] length in chars
                                            must be an even number         */

/********************** ClassArray[classtypes][] ************************/

#define ExternalEvent            0       /* index for cmu use only         */
#define InternalEvent            1       /* index for unc and cmu use      */

/********************** InfoArray[infoitems] ****************************/

#define NumErrors        0               /* index of number of errors seen */
#define SysMonReturn     1               /* index of monitor return value  */

/************************ Trace values **********************************/

#define NoTrace                  0       /* completely quiet               */
#define HighLevelTrace           10      /* general control flow           */
#define LowLevelTrace            100     /* detailed control flow          */
#define DemoATrace               5       /* Prints a * each time a packet is*/
                                         /* sent, and a # each time a packet*/
                                         /* is resent due to an ethernet   */
                                         /* transmission error             */
#define UNCDEMO                  1       /* for running at UNC             */

/************************ Disk  values **********************************/
#define PMode                    0644    /* rw_r__r__ protection           */
#define File_Name_Prefix "sendata.XXXXXX" /* prefix for disk output file   */
#ifndef MaxFileSize
#define MaxFileSize      4096*(1024+12)
#endif  MaxFileSize
/*
 * Defines the maximum number of chars before switching files.  This is
 * determined by speed of access and local disk usage.  For the fastest
 * access, only the direct pointers of the inode are used (12 of them),
```

```
* each of which points to 4K bytes.  An alternate value would be using
* the indirect blocks, the first of which points to 1024 block pointers,
* yielding 4Mb + 48Kb as the file size.  The accountant on a Sun 2 was
* able to produce 50,000 bytes in 3 seconds, and a minutes worth of data
* minimum should be in a single file.
*/
```

---

vars.h

```
/*****************************************************************************/
/*                              vars.h                                       */
/* This file contains the definitions of all globals used by the accountant.*/
/*                                                                           */
/* Original author: Richard Snodgrass                                        */
/* Modifications for unc implementation: Steven Reuman                       */
/*****************************************************************************/

unsigned int                    /* The actual user parts of the messages     */
        cmdbuffer[12],
        databuffer[256];
#if STANDALONE
        char    *File_Name;     /* disk file for output                      */
        int     disk_dump;      /* file discriptor for File_name             */
        char    *mktemp();      /* used to create unique file names          */
#endif

/*********************** Structured Variables ********************************/
/* Arrays to record active processes, sensors, and event records.            */
/* In the UNC implementation, the following definitions are relevant:        */
/*                                                                           */
/*      1) class - all the information in one sensor description file.       */
/*               - set of objects                                            */
/*               - set of events                                             */
/*                 In the UNC implementation, there is only one class.       */
/*      2) object- process-id                                                */
/*      3) event - all the information in one event record                   */
/*               - set of domains                                            */
/*               - output of one sensor in the target program                */
/*      4) domain- datatype output by sensor in target program               */
/*               - integer, double, string, char, etc.                       */
/*                                                                           */
/* Note that ClassArray[][].EventID[x] - unique event-id so we could index   */
/* with the following                                                        */
/*      EventArray[ ClassArray[][].EventID[x] ]                              */
/* Note that ClassArray[][].EventID[x] and EventArray[x] are indexed by the  */
/* same unique event-id for a given sensor.  The two data structures have    */
/* been separated to enable simpler commands to be sent to the accountant    */
/* (i.e., this way, no ClassType (ClassArray[x][]) or ClassNumber            */
/* (ClassArray[][x]) has to be sent in command in order to AddDomainToEvent  */
/* for example).                                                             */
/*****************************************************************************/

struct ClassRecord {
        int NumSNames;          /*index of ClassArray[][].SName[x]          */
        int NumEvents;          /*index of ClassArray[][].EventID[x]        */
        int SName[NumObjectsPerClass];  /*list of object-ids in the class   */
        char position[NumEventsPerObject];/*sensor bit position in cmd buffer*
/
        char EventID[NumEventsPerClass];/*list of event-ids in the class     */
        } ClassArray[MaxClassTypes][MaxNumClasses];

struct EvntRecord {
        char TimeStamp;         /*1=enabled (get time of event); 0=disabled */
        char NumDomains;        /*index into EventArray[].Domains[x]        */
        char Domains [MaxNumDomains];   /*list of domains in event record   */
        } EventArray[MaxNumEvents];     /* indexed by unique event id       */

unsigned int
        InfoArray[MaxInfoItems];/*data stored by accountant for monitor     */
unsigned short
        EventRecBuf[MaxEventRecordLength/2];/*interface with sensorread in
                                              shorts--unc data structure used
                                              comparably to pup.data[] of
                                              cmu implementation            */

/********************** Simple variables *************************************/
char *invocationcomment;

int     InitCount,              /*initial number of commands received       */
        trace,                  /*hi=little printout; low=lots of printout  */
        NumEvents;              /*number of events currently declared to acc*/
unsigned int
        filltime,               /*timeout used to wait for the buffer to fill,
                                  in seconds*/
        GlobalTimeStamp;        /*generation time of last event record
                                  picked up from kernel                     */
char    *sensor_invocation;
```

accountant.sen

```
-- Sensor description file for the accountant, version 2
-- Rick Snodgrass, May 29, 1984

User Taskforce Accountant is

Process UnixAccountant is

IncludeFileName "accountant.h";

Event Restart (time, accountanttime, kerneltime: doubleinteger;
               HostName: String[16];
               AsciiTime: String[16];  -- should be the same info as time(2)
               Invocation: String[80]) -- invocation line, with comment
       is timestamped, sensortraced, assumedenabled;

Event Status (Users, Load, Running, Blocked, Swapped: integer) -- all counts
       is timestamped, sensortraced, assumedenabled;

end UnixAccountant;

end Accountant.
```

accountant.h

```
#include <monitor/montypes.h>
#include <monitor/mondefs.h>
#include <netinet/in.h>
#include <sys/syslocal.h>
short u__boolvec[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,};
mon_putevent u__sbuffer;
#define  RestartSensor(thistime,accttime,kerntime,hostname,chartime,inittext)\
{                                                                            \
        register mon_putevent  *u__reg_buf = &u__sbuffer;                    \
        register short  *u_sen_fields = (short *)u__reg_buf->fields;         \
        register mon_string u_sen_f_ptr = { mon_string }hostname;            \
        register mon_string u_sen_f_end = u_sen_f_ptr+12*2/sizeof(mon_string);\
        register short  u_reg_length;                                        \
        u__reg_buf->eventnumber = 0;                                         \
        u__reg_buf->object      = 0;                                         \
        u__reg_buf->timestamp   = 1;                                         \
        *(long *)u_sen_fields   = (long)thistime;                            \
        u_sen_fields           += 2;                                         \
        *(long *)u_sen_fields   = (long)accttime;                            \
        u_sen_fields           += 2;                                         \
        *(long *)u_sen_fields   = (long)kerntime;                            \
        u_sen_fields           += 2;                                         \
        do { *u_sen_fields++ = PackStr(u_sen_f_ptr); }                       \
        while  ( NotEOS(u_sen_f_ptr, u_sen_f_end) );                         \
        *(u_sen_fields - 1) &= ntohs(0xff00);                                \
        u_sen_f_ptr  =  (mon_string) chartime;                               \
        u_sen_f_end  =  u_sen_f_ptr + 27*2/sizeof(mon_string);               \
        do { *u_sen_fields++ = PackStr(u_sen_f_ptr); }                       \
        while  ( NotEOS(u_sen_f_ptr, u_sen_f_end) );                         \
        *(u_sen_fields - 1) &= ntohs(0xff00);                                \
        u_sen_f_ptr  =  (mon_string) inittext;                               \
        u_sen_f_end  =  u_sen_f_ptr + 127*2/sizeof(mon_string);              \
        do { *u_sen_fields++ = PackStr(u_sen_f_ptr); }                       \
        while  ( NotEOS(u_sen_f_ptr, u_sen_f_end) );                         \
        *(u_sen_fields - 1) &= ntohs(0xff00);                                \
        u_reg_length  =  u_sen_fields - (short *)u__reg_buf;                 \
        u__reg_buf->cmd.type  =  MONOP_PUTEVENT_EXT;                         \
        u__reg_buf->cmd.length  =  u_reg_length;                             \
        syscall(SYSLOCAL, SYSL_MONITOR, ( unsigned char * )&u__sbuffer);\
}

#define   Status(users, load, running, blocked, swapped)                     \
{                                                                            \
        register  mon_putevent  *u__reg_buf = &u__sbuffer;                   \
        register short  * u_sen_fields = (short *)u__reg_buf->fields;        \
        register short   u_reg_length;                                       \
        u__reg_buf->cmd.type = MONOP_PUTEVENT_EXT;                           \
        u__reg_buf->cmd.length  = u_sen_fields - u__reg_buf +5;              \
        u__reg_buf->eventnumber = 1;                                         \
        u__reg_buf->object      = 0;                                         \
        u__reg_buf->timestamp   = 1;                                         \
        *u_sen_fields++ = users;                                             \
        *u_sen_fields++ = load;                                              \
        *u_sen_fields++ = running;                                           \
        *u_sen_fields++ = blocked;                                           \
        *u_sen_fields++ = swapped;                                           \
        syscall(SYSLOCAL, SYSL_MONITOR, (unsigned char *)&u__sbuffer);  \
}
```

main.c

```
/*****************************************************************************/
/*                          ACCOUNTANT                                       */
/*                                                                           */
/* Original author: Richard Snodgrass                                        */
/* Modifications for UNC implementation: Steven Reuman                       */
/*                                       Stephen Duncan                      */
/*                                                                           */
/* General organization:                                                     */
/*      Main procedure                                                       */
/*      InitAcc                                                              */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/

#ifndef lint
static char rcsheader[] = "$Header: main.c,v 1.3 85/11/13 00:16:12 duncans Exp
 $";
#endif lint

/************************** Define Implementation ***************************/
/*                                                                           */
/* These are flags to the compiler:                                          */
/* UNC          1               def = unc, ndef = cmu                        */
/* STANDALONE   1               def = standalone mode                        */
/*                              ndef = in monitor                            */
/* MONITOR      1               must be defined to enable monitoring         */
/*                                                                           */
/* These are set in const.h                                                  */
/* UNCDEMO      1               trace level need to demo                     */
/*****************************************************************************/

#include <sys/types.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <stdio.h>
#include <signal.h>
#include <monitor/monops.h>
#include <monitor/monerrcds.h>
#include "accountant.h"         /* sensors */
#include "const.h"
#include "vars.h"

#ifndef UNC
#ifdef ATCMU
#include <sys/ipc.h>
#else
#include "ipc.h"
#endif ATCMU
#include "queue.h"
#include "enet.h"
#include "CMUconst.h"
#include "CMUvars.h"
#include "CMU.c"
#endif  /* #if !UNC */


/************************ Structured Variables ******************************/
/* Arrays to record active processes, sensors, and event records.            */
/* In the UNC implementation, the following definitions are relevant:        */
/*                                                                           */
/*      1) class = all the information in one sensor description file.       */
/*                 = set of objects                                          */
/*                 = set of events                                           */
/*                   In the UNC implementation, there is only one class.     */
/*      2) object= process-id                                                */
/*      3) event = all the information in one event record                   */
/*                 = set of domains                                          */
/*                 = output of one sensor in the target program              */
/*      4) domain= datatype output by sensor in target program               */
/*                 = integer, double, string, char, etc.                     */
/*                                                                           */
/*****************************************************************************/

unsigned int
        InfoArray[MaxInfoItems];/*data stored by accountant for monitor     */
unsigned short
        EventRecBuf[MaxEventRecordLength/2];/*interface with sensorread in
                                              shorts--unc data structure used
                                              comparably to pup.data[] of
                                              cmu implementation            */

/************************ Simple variables **********************************/
char *invocationcomment;

int     InitCount,              /*initial number of commands received       */
        trace,                  /*hi=little printout; low=lots of printout  */
        NumEvents;              /*number of events currently declared to acc*/
unsigned int
        filltime,               /*timeout used to wait for the buffer to fill,
                                  in seconds*/
        GlobalTimeStamp;        /*generation time of last event record
                                  picked up from kernel                     */
char    *sensor_invocation;
```

```
unsigned int                    /* The actual user parts of the messages    */
        cmdbuffer[12],
        databuffer[256];

char    AccountantName[100];
char    KernelName[100];
char    InitialText[100];
int     KernelVectorSize = 0;
short   ActiveSensors[] = { 1, 2, 3, 5, 6, 7, 8, 9, 0 };
                                /* sizeof is used to get the num. of elem. */



/*************************************************************************/
/*      MAIN                                                             */
/* This procedure is the driver routine for the accountant (cmu and unc  */
/* implementations).  Its major functions are to initialize the accountant*/
/* in InitAcc() and execute an infinite loop of                          */
/*                                                                       */
/*              1) receive and execute a command sent by the monitor by  */
/*                 calling Processcmd()                                  */
/*              2) handle communication with the target program running  */
/*                 on Cm* (cmu) or vax (unc) by calling                  */
/*                      DoProtocol()                                     */
/*                   or DoUnixProto()                                    */
/*                                                                       */
/* Original author: Richard Snodgrass                                    */
/* Modifications for unc implementation: Steven Reuman, Steve Duncan     */
/*************************************************************************/

main (argc, argv)
int argc;
char *argv[];
{
    int i;

#ifdef STANDALONE
    int Finish();                       /*closedown routine             */
    signal(SIGTERM,Finish);             /*exit from while if STANDALONE */
#endif

/* Give defaults for switch values */
    strcpy(AccountantName, "accountant");
    strcpy(KernelName, "/vmunix.monitor");
    strcpy(InitialText, "Accountant started");
    trace = 0;

/* Get switches */
    for (i = 1; i < argc; i++)
        {
        if (strcmp(argv[i], "-a") == 0 && ++i < argc)
                {
                strncpy(AccountantName, argv[i], sizeof(AccountantName));
                continue;
                }

        if (strcmp(argv[i], "-k") == 0 && ++i < argc)
                {
                strncpy(KernelName, argv[i], sizeof(KernelName));
                continue;
                }

        if (strcmp(argv[i], "-t") == 0 && ++i < argc)
                {
                strncpy(InitialText, argv[i], sizeof(InitialText));
                while ( (i+1) < argc && *argv[i+1] != '-')
                        {
                        if (strlen(InitialText) + 1 < sizeof(InitialText))
                                strcat(InitialText, " ");
                        strncat(InitialText, argv[++i], sizeof(InitialText) -
strlen(InitialText) - 1);
                        }
                continue;
                }

        if (strcmp(argv[i], "-x") == 0 && ++i < argc)
                {
                trace = atoi(argv[i]);
                continue;
                }

        printf("accountant: usage [-a accountantname] [-k kernelname] [-t init
ialtext] [-x tracelevel]\n");
        exit(-1);
        }

    InitAcc ();                         /*initialize the accountant       */
    while (1)                           /*get commands and process        */
        {                               /*InitCount commands expected     */
#ifdef STANDALONE
        sleep(filltime);                        /* wait for event buffer to fi
ll  */
#else
        for (i = 1; i <= InitCount; i++)
            if (ipcreceive (cmdport, &cmdmsg, InitTime)==1) Processcmd();
#endif  /* #else not STANDALONE */

#ifndef UNC
                DoProtocol ();          /*get Cm* response,send to monitor*/
#else
                DoUnixProto();          /*get event record (unc), send    */
                if (trace == ~0) Finish();
```

```
#endif /* else of #if !UNC */
        }
}


/************************************************************************/
/*      Initacc()                                                       */
/*                                                                      */
/* This is the initialization procedure for the accountant.  In the CMU */
/* implementation, it initializes the connection between Cm* and the    */
/* accountant, sends an initial acknowledgement message to the monitor  */
/* and checks for the first packet of data from the target program(s).  */
/* All monitor commands received before the                             */
/* StartNormalProcessing command are simply routed to Cm*.  In the UNC  */
/* implementation, connection is established with kernel memory and     */
/* the kernel event record buffer cleared for communication with the    */
/* target program.  The following serial functions are performed        */
/* and commented as below:                                              */
/*                                                                      */
/*      1. initialize ipc data structures by calling initipc()          */
/*      2. if operating on Cm*, establish connection with Cm*           */
/*      3. if operating on UNC, establish connection with kernel        */
/*      4. when successful, send acknowledgement to monitor             */
/*      5. initialize simple and structured global variables            */
/*      6. wait for the StartNormalProcessing command.                  */
/*      7. get initial data message from Cm* or target program          */
/*         to monitor by calling doprotocol() or dounixproto().         */
/*                                                                      */
/* Original author: Richard Snodgrass                                   */
/* Modifications for UNC implementation: Steven Reuman                  */
/************************************************************************/

InitAcc ()
{
    int     number,
            i,j,
            ClassType,
            finished;
    char    namestring[15];
    long    sensor_thistime,
                sensor_accountanttime,
                sensor_kerneltime;
    char    sensor_hostname[255];
    char    *ascii_time;
    struct stat stat_buf;
    short   cmd[6];             /*buffer for command message to kernel  */
#ifdef UNC
    struct mon_cmd command;     /* holds command for kernel */
#endif UNC

/*      1. initialize ipc data structures by calling initipc()          */
```



```
/*          or by opening disk file when STANDALONE                     */

#ifdef STANDALONE
    InitOutput();               /* Creates file for output */
#else
    InitIPC ();
#endif

#ifndef UNC
/*      2. if operating on Cm*, establish connection with Cm*           */

        for (i=0; i<=15; i++)
                {
                sprintf (namestring, "/dev/enet%d", i);
                EnetDesc = open (namestring, 2);
                if (trace >= LowLevelTrace)
                    printf ("%s -> %d\n", namestring, EnetDesc);
                if (EnetDesc != -1)
                    break;
                }

            ioctl (EnetDesc, EIOCSETF, &myfilter);

            sendpkt[0].pup.chksum = 0177777;
            sendpkt[0].pup.pup_type = MonDataPup;

            sendpkt[1].pup.chksum = 0177777;
            sendpkt[1].pup.pup_type = MonDataPup;

            settimeout (EnetDesc, TimeOut);

                /* write initial data message to Cm*; get response in   */
                /* recvpkt.  "number" = number of bytes returned--not   */
                /* currently used.                                      */
            while (1)
                {
                write (EnetDesc, sendpkt, PAKETMINLENGTH);
                number = loread (EnetDesc, &recvpkt, PAKETLENGTH);
                if ((recvpkt.pup.pup_type == MonAckPup) &&
                        (recvpkt.pup.pup_id[1] == 0))
                    break;
                }

            databuffer[0] = InitDataRecord;
            for (i=1; i<=InitDataWords;i++)
                {
                databuffer[i] = (recvpkt.pup.data[i * 2 - 2] << 16);
                databuffer[i] += recvpkt.pup.data[i * 2 - 1];
                }

            if (trace >= HighLevelTrace)
```

```
            printf ("Connection established on Cm*: [2]:%d\n", databuffer[2]
);
        /* end non-UNC section */

#else   /* UNC */
/*      3. if operating on UNC, establish connection with kernel        */
/* For the UNC implementation, the monitor(9) call used below opens and */
/* clears the kernel file to be shared with the target program.  It     */
/* returns the size in bytes of the e.r. buffer allocated.  This        */
/* is then passed back to the monitor in an InitDataRecord.             */

            if (trace==UNCDEMO) printf("\n[ACC: Locating Kernel Storage...]\n"
);
            /*
             * Start up kernel monitor
             */
            command.type = MONOP_INIT;
            command.length  = (sizeof(struct mon_cmd)+1)/2;
            KernelVectorSize = syscall(SYSLOCAL,SYSL_MONITOR,
                (u_char *)&command);
            databuffer[0] = InitDataRecord;
            databuffer[1] = (unsigned int)KernelVectorSize;
            databuffer[2] = 0;
            databuffer[3] = 0;
            if (trace >= HighLevelTrace && databuffer[1]>0)
                {
                printf ("InitAcc: Kernel connection established:\n");
                printf ("Shared memory available = %d bytes\n", databuffer[1])
;
                }
            /*
             * Get data for RestartSensor
             */
            sensor_thistime = time(0);
            ascii_time = ctime( &sensor_thistime );
            gethostname(sensor_hostname, 80);
            if (stat(AccountantName, &stat_buf)==0)
                sensor_accountanttime = stat_buf.st_mtime;
            else sensor_accountanttime = -1;
            gethostname(sensor_hostname, 80);
            if (stat(KernelName, &stat_buf)==0)
                sensor_kerneltime = stat_buf.st_mtime;
            else sensor_kerneltime = -1;
            RestartSensor(sensor_thistime, sensor_accountanttime,
                sensor_kerneltime, sensor_hostname, ascii_time,
                InitialText);
#ifdef STANDALONE
            /*
             * Note that ActiveSensors is null terminated
             * (null is not a valid sensor id)
             */
            TurnOnSensors( ActiveSensors );
#endif


            if (trace == UNCDEMO)
                printf ("\n[ACC: Kernel Connection established -- ");
            if (trace == UNCDEMO)
                printf ("%d bytes available.]\n", databuffer[1]);
#endif #else

#ifndef STANDALONE
/*      4. when successful, send acknowledgement to monitor             */
    if (trace >= LowLevelTrace)
        printf("InitAcc: calling Senddata to ack open shared memory\n");
    SendData (InitDataWords + 1);
#endif

/*      5. initialize simple and structured global variables            */

    GlobalTimeStamp = 0;
    filltime = 1;               /* in seconds */
#ifndef UNC
    CurrPup = &sendpkt[1];
    SeqNum = 0;
    StarMonUsedWords = 0;
    CurrentSend = 1;
    CurrPtr = 0;
#endif

    for (ClassType = 0; ClassType < MaxClassTypes; ClassType++)
        for (i = 0; i < MaxNumClasses; i++)
            {
            ClassArray[ClassType][i].NumSNames = 0;
            ClassArray[ClassType][i].NumEvents = 0;
            for(j=0;j<NumEventsPerClass;j++)
                    ClassArray[ClassType][i].position[j] = 0;
            }
    NumEvents = 0;
    for (i = 0; i < MaxNumEvents; i++) EventArray[i].NumDomains = 0;

#ifndef STANDALONE
/*      6. wait for the StartNormalProcessing command.                  */
/* In the CMU implementation, if other commands are received beforehand */
/* they are passed on to Cm*, but should not fill up more than one      */
/* packet to the resident monitor.  In the UNC implementation, other    */
/* commands are ignored.                                                */

   if (trace >= LowLevelTrace)
        printf("Initacc: calling ipcreceive for StartNormalProcessing\n");

   finished = 0;
```

```
   while (!finished)
        {
        if (ipcreceive (cmdport, &cmdmsg, InitTime) == 1)
                {
                if (trace > HighLevelTrace)
                    printf("InitAcc: command received: %d\n",cmdbuffer[0]);
                if (cmdbuffer[0] == StartNormalProcessing) finished = 1;
#ifndef UNC
                Processcmd();
#endif
                }
        }
#endif /* !STANDALONE */

   if (trace >= HighLevelTrace)
        printf ("InitAcc: Initialization commands have been processed.\n");

/*      7. get initial data message from Cm* or target program          */
/*         to monitor by calling doprotocol() or dounixproto().         */

#ifndef UNC
        DoProtocol();           /* retrieve StarMon sensor description  */
#else
        DoUnixProto();          /* check for event records              */
        if (UNCDEMO) printf("\n[ACC: Target program initialization record now
available.]\n");
#endif
}       /* InitAcc */
```

DoUnixProto.c

```
/*****************************************************************************/
/*      DoUnixProto()                                                        */
/* This routine is the back part of the processcmd() loop.  After each  */
/* command is processed, control flows here so the accountant can       */
/* continue to pick up event records from the event record buffer.      */
/* If the event record buffer is not empty (return value = 2) the       */
/* the data retrieved is sent to the monitor by calling                 */
/* SendDataRecords().  The routine is UNC implementation specific.      */
/*                                                                      */
/* Author: Steven Reuman                                                */
/*****************************************************************************/

#include <monitor/monops.h>
#include <monitor/montypes.h>
#include <sys/syslocal.h>
#include "const.h"

extern int      KernelVectorSize;
extern short    EventRecBuf[];
extern int      trace;

DoUnixProto()
{
        mon_getevent    GetEvent;
        int             count;

        GetEvent.cmd.type     = MONOP_GETEVENTS;
        GetEvent.cmd.length   = (sizeof (mon_getevent)+1) /2;
        GetEvent.req_length   = KernelVectorSize;
        GetEvent.acct_buf_ptr = EventRecBuf;
        if ( (count = syscall(SYSLOCAL,
                        SYSL_MONITOR,
                        (unsigned char *)&GetEvent)
             ) > 0)
        {
#ifdef STANDALONE
                WriteEventRecord ((sizeof(short) * count),EventRecBuf);
#ifdef UNCDEMO
                if (trace == UNCDEMO)
                {
                    /* print out the first event record's fixed fields */
                    mon_putevent        *pevt = (mon_putevent *)EventRecBuf;

                    printf("%-5.4u",    pevt->cmd.type);
                    printf("%-5.4u",    pevt->cmd.length);
                    printf("%-10.8u",   pevt->eventnumber );
                    printf("%-10.8u",   pevt->performer );
                    printf("%-10.8u",   pevt->object );
                    printf("%-10.8u",   pevt->object );
                    printf("%-10.8u",   pevt->initiator );
                }
#endif
#else
                SendDataRecords();      /*write to output*/
#endif
        }
}
```

Finish.c

```
#include <stdio.h>
#include <monitor/monops.h>
#include <monitor/montypes.h>
#include <sys/syslocal.h>
#include "const.h"

extern int      disk_dump;              /* output file of event records */
extern int      ActiveSensors[];        /* array of active sensors      */
extern int      trace;

/*****************************************************************************/
/* Finish                                                                    */
/*      This is the exit routine for STANDALONE mode, reached via            */
/*      the sigset call.  It turns off all sensors, gets the last            */
/*      data from the event buffer, and closes the disk file.                */
/*                                                                           */
/* Stephen Duncan                                                            */
/*****************************************************************************/

Finish()
{
        struct mon_cmd  command;

        if (trace >= LowLevelTrace)
        {
                fprintf(stderr,"entered Finish()\n");
                fflush(stderr);
        }

        TurnOffSensors(ActiveSensors);  /* stop recording */
        DoUnixProto();                  /* Get last events */

        command.type = MONOP_SHUTDOWN;  /* Close down system */
        command.length = (sizeof(struct mon_cmd)+1)/2;
        syscall(SYSLOCAL, SYSL_MONITOR, (unsigned char *)&command);

        close(disk_dump);
        exit(StdTermination);
}

/*****************************************************************************/
/* AbortAcct                                                                 */
/*      Disable the kernel sensors, shutdown accounting, and exit.           */
/*                                                                           */
/* Stephen Duncan                                                            */
/*****************************************************************************/

AbortAcct(error)
int     error;                          /* exit code */
{
        struct mon_cmd  command;

        TurnOffSensors(ActiveSensors);

        command.type  = MONOP_SHUTDOWN;
        command.length = (sizeof(struct mon_cmd)+1)/2;
        syscall(SYSLOCAL, SYSL_MONITOR, (unsigned char *)&command);

        exit(error);
}
```

WriteEventRecord.c

```
#ifndef lint
static char *rcsheader = "$Header: WriteEventRecord.c,v 1.3 85/11/13 00:15:25
duncans Exp $";
#endif lint

/*****************************************************************************/
/* These routines control the output to files for the UNC                    */
/* implementation.                                                           */
/*      WriteEventRecord                                                     */
/*      InitOutput                                                           */
/*      SwitchFiles                                                          */
/*      GenTemplate                                                          */
/* If a critical failure occurs, AbortAcct is called to clean things up      */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <ctype.h>
#include "const.h"


extern int      trace;                  /* indicates trace level        */
static int      disk_dump;              /* file descriptor of output    */
static char     *File_Name;             /* current output file          */
```

```
static int      FileSize = 0;           /* to det. when to switch files */
static int      a_char = -1, b_char = 0; /* for GenTemplate             */
static char     char_list[] =
        "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";


/*****************************************************************************/
/* WriteEventRecord                                                          */
/*      Write out a record to a disk file in STANDALONE mode.                */
/*      This is UNIX specific.                                               */
/*                                                                           */
/* Stephen Duncan                                                            */
/*****************************************************************************/

WriteEventRecord (count,data)           /* puts to disk */
int     count;
char    *data;
{
        if ( count + FileSize > MaxFileSize ) SwitchFiles();
        if ( write(disk_dump,data,count) < 0 )
        {
                perror("WriteEventRecord: writing disk_dump");
                AbortAcct(CantWrite);   /* Need to shut down things */
        }
        FileSize += count;
}

/*****************************************************************************/
/* InitOutput                                                                */
/*      Open the first file for output.                                      */
/*                                                                           */
/*                                                                           */
/* Stephen Duncan                                                            */
/*****************************************************************************/
InitOutput()
{
        char    *template;
        char    *mktemp();
        char    *GenTemplate();

        template = GenTemplate(File_Name_Prefix);
        File_Name = mktemp(template);
        if ((disk_dump = creat(File_Name,PMode)) < 0)
        {
                fprintf(stderr,"creat failed for %s \n",File_Name);
                AbortAcct(CantOpen);
        }
        else if (trace >= HighLevelTrace)
                printf("File %s created\n",File_Name);
}
/*****************************************************************************/
```



```
/* SwitchFiles                                                               */
/*      Close disk_dump, create a new unique file, dup disk_dump.            */
/*                                                                           */
/* Stephen Duncan                                                            */
/*****************************************************************************/
static
SwitchFiles()
{
        char    *template;
        char    *mktemp();

        if ( close(disk_dump) < 0 )
        {
                perror("SwitchFiles: closing disk_dump");
                AbortAcct(BadClose);    /* Need to close down accountant */
        }
        template = GenTemplate(File_Name_Prefix);
        File_Name = mktemp(template);
        if ( (disk_dump = creat(File_Name,PMode)) < 0 )
        {
                perror("SwitchFiles: creating new file");
                AbortAcct(BadClose);
        }
        FileSize = 0;
}
/*****************************************************************************/
/* GenTemplate                                                               */
/*      Generate a new template for the files from a constant string.        */
/*                                                                           */
/* Stephen Duncan                                                            */
/*****************************************************************************/
static char *
GenTemplate(template)
char    *template;
{
        /*
         * Form of template is [.]*xxxxxx
         * Note that static variables a_char and b_char are used to
         * retain info between calls.  It doesn't really matter what they
         * are initially.
         */

        int     len;                    /* length of template */
        char    *new;                   /* new template */

        if ( (len = strlen(template)) <= 7 )
        {
                fprintf(stderr, "GenTemplate: invalid template: %s\n",
                        template);
                AbortAcct(CantOpen);
        }
```

```
        new = (char *)malloc( len+1 );
        strcpy(new,template);

        /*
         * The following section changes two postions of the template
         * to ease the work of mktemp.  When a_char reaches the end,
         * it resets to 0 and increments b_char.  This yields 36^2
         * unique templates.  mktemp() is still used to check for
         * file names and to add the pid.  It can only handle 26
         * files, however.
         */
        if (char_list[a_char+1] == 0)
        {
                a_char =  0;
                if (char_list[b_char+1] == 0)
                        b_char = 0;
                else
                        b_char++;
        }
        else
                a_char++;

        new[len-7] = char_list[a_char];
        new[len-8] = char_list[b_char];

        return(new);
}
```

SensorControl.c

```
#ifndef lint
static char *rcsheader = "$Header: SensorControl.c,v 1.3 85/11/13 00:15:06 dun
cans Exp $";
#endif lint

#include <stdio.h>
#include <sys/syslocal.h>
#include <monitor/monops.h>
#include <monitor/montypes.h>

/* Commands to sysl_monitor to enable/disable sensors */

static mon_putreq off_preq = {
        { (char)MONOP_PUTREQ,
          (char)((sizeof(mon_putreq)+1)/2) },   /* type,length  */
        { 0, 0, 0 }                             /* pid, event, enablevalue      */
        };
static mon_putreq on_preq = {
        { (char)MONOP_PUTREQ,
          (char)((sizeof(mon_putreq)+1)/2) },   /* type,length  */
        { 0, 0, 1 }                             /* pid, event, enablevalue      */
        };

/*****************************************************************************/
/* TurnOffSensors                                                            */
/*      Turn off all active sensors                                          */
/*                                                                           */
/* Stephen Duncan                                                            */
/*****************************************************************************/

TurnOffSensors(ActiveSensors)
short   ActiveSensors[];                /* Null terminated array */
{
    int i;                              /* subscript for ActiceSensors */

    for (i = 0; ActiveSensors[i]; i++)
    {
        off_preq.req.eventnumber = ActiveSensors[i];
        syscall(SYSLOCAL, SYSL_MONITOR, (unsigned char *)&off_preq);
    }
}
/*****************************************************************************/
/* TurnOnSensors                                                             */
/*      Turn on sensors in ActiveSensors                                     */
/*                                                                           */
/* Stephen Duncan                                                            */
/*****************************************************************************/
TurnOnSensors(ActiveSensors)
short   ActiveSensors[];                /* Null terminated array */
{
        int             i,status;

        for (i = 0; ActiveSensors[i]; i++)
        {
                on_preq.req.eventnumber = ActiveSensors[i];
                status = syscall(SYSLOCAL,SYSL_MONITOR,
                    (unsigned char *)&on_preq);
                if ( status < 0 )

                    fprintf(stderr,
                        "sysl_monitor enable failed for %d with code %d\n",
                        on_preq.req.eventnumber, status);
        }
}
```

ACCT

makefile

```
MONDEF = MONITOR
MONINCLUDE = monitor
DEBUG = -DMONDEBUG -DDEBUG
CFLAGS = -D$(MONDEF) -Usun

acct:   acct.c ../$(MONINCLUDE)/montypes.h acct_sensors.h
        cc  -D$(MONDEF) -o acct acct.c
```

acct_sensors.h

```
#include "../monitor/montypes.h"
#include "../monitor/mondefs.h"
#include <netinet/in.h>
#include <sys/syslocal.h>
short u__boolvec[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,};
mon_putevent u__sbuffer;
#define  AcctHeader(acctname, kernelname, init_text)  \
{  \
        register mon_putevent  *u__reg_buf = &u__sbuffer;  \
        char     namespace[26]; \
        char     *asciitime, *ctime(); \
        long     ltime = time(); \
        register short  *u_sen_fields = (short *)u__reg_buf->fields;  \
        register mon_string u_sen_f_ptr = ( mon_string )namespace; \
        register mon_string u_sen_f_end = u_sen_f_ptr + 5*2/sizeof(mon_string)
;  \
        register short  u_reg_length; \
        struct   stat u_sen_stat; \
        u__reg_buf->eventnumber = 0; \
        u__reg_buf->object = 0; \
        u__reg_buf->timestamp = 1; \
        stat(acctname,&u_sen_stat); \
        *(long *)u_sen_fields = (long)u_sen_stat.st_mtime; \
        u_sen_fields += 2; \
        stat(kernelname,&u_sen_stat); \
        *(long *)u_sen_fields = (long)u_sen_stat.st_mtime; \
        u_sen_fields += 2; \
        gethostname(namespace, 12); \
        do { *u_sen_fields++ = PackStr(u_sen_f_ptr); }  \
        while  ( NotEOS(u_sen_f_ptr, u_sen_f_end) );  \
        *(u_sen_fields - 1) &= ntohs(0xff00);  \
        asciitime = ctime(&ltime); \
        u_sen_f_ptr = (mon_string) asciitime;  \
        u_sen_f_end = u_sen_f_ptr + 12*2/sizeof(mon_string); \
        do { *u_sen_fields++ = PackStr(u_sen_f_ptr); }  \
        while  ( NotEOS(u_sen_f_ptr, u_sen_f_end) );  \
        *(u_sen_fields - 1) &= ntohs(0xff00);  \
        u_sen_f_ptr = (mon_string) init_text; \
        u_sen_f_end = u_sen_f_ptr + 127*2/sizeof(mon_string); \
        do { *u_sen_fields++ = PackStr(u_sen_f_ptr); }  \
        while  ( NotEOS(u_sen_f_ptr, u_sen_f_end) );  \
        *(u_sen_fields - 1) &= ntohs(0xff00);  \
        u_reg_length = u_sen_fields - (short *)u__reg_buf;  \
        u__reg_buf->cmd.type = MONOP_PUTEVENT_EXT; \
        u__reg_buf->cmd.length = u_reg_length; \
        syscall(SYSLOCAL, SYSL_MONITOR, ( unsigned char * )&u__sbuffer);  \
}
#define   UserXamplSensor(obj, str_parm_1, sh_parm_2, str_parm_3, lg_parm_4) \
if   ( u__boolvec[0] & 0x1<<i )   \
{  \
        register  mon_putevent  *u__reg_buf = &u__sbuffer;   \
        register short  * u_sen_fields = (short *)u__reg_buf->fields;   \
        register mon_string u_sen_f_ptr = (mon_string)str_parm_1;   \
        register mon_string u_sen_f_end = u_sen_f_ptr + 127*2/sizeof(mon_strin
g);  \
        register short   u_reg_length;   \
        u__reg_buf->eventnumber = 1;  \
        u__reg_buf->object = obj;  \
        u__reg_buf->timestamp = 1;  \
        do { *u_sen_fields++ = PackStr(u_sen_f_ptr); }  \
        while  ( NotEOS(u_sen_f_ptr, u_sen_f_end) );  \
        *(u_sen_fields - 1) &= ntohs(0xff00);   \
        *u_sen_fields++ = sh_parm_2;   \
        u_sen_f_ptr = (mon_string) str_parm_3; \
        u_sen_f_end = u_sen_f_ptr + 127*2/sizeof(mon_string); \
        do { *u_sen_fields++ = PackStr(u_sen_f_ptr); }  \
        while  ( NotEOS(u_sen_f_ptr, u_sen_f_end) );  \
        *(u_sen_fields -1) &= ntohs(0xff00);   \
        *(long *)u_sen_fields = lg_parm_4;   \
        u_sen_fields += 2;   \
        u_reg_length = u_sen_fields - (short *)u__reg_buf;   \
        u__reg_buf->cmd.type = MONOP_PUTEVENT_EXT;  \
        u__reg_buf->cmd.length = u_reg_length;  \
        syscall(SYSLOCAL, SYSL_MONITOR, (unsigned char *)&u__sbuffer);   \
```

```
}
```

acct.c

```
#include <sys/param.h>
#include <sys/dir.h>
#include <sys/systm.h>
#include <sys/user.h>
#include <sys/proc.h>
#include <sys/stat.h>
#include "acct_sensors.h"
#include <stdio.h>
#include <ctype.h>
#define ALL


short           buffer[512*50];
main (argc,argv)
int     argc;
char    *argv[];
{
        int     runminutes,             /* accumulated run time */
                runduration,            /* duration of event gathering */
                sleeptime;              /* duration of sleep */
        int     status;                 /* status of calls */
        int     i;
        char    *numeric;               /* for testing arg */
        short   charcvt[12],charcvt2[12]; /* the sun requires alignment */

        fprintf(stderr,"argc:%d\n",argc);
        switch(argc)                    /* determine run duration */
        {
                case 1: runduration = 20;
                        break;
                case 2: /*for ( numeric=argv[1] ; *numeric ; numeric++ )
                                if ( !isdigit(*numeric) )
                                {
                                        fprintf(stderr,
                                                "%s: arg must be an integer\n"

                                                argv[0]);
                                        exit(1);
                                }
                                */
                                sscanf(argv[1],"%u",&runduration);
                                break;
                        default:
                                fprintf(stderr,"%s: invalid number of args\n",argv[0])
;
                                exit(1);
        }

        if ( (status=startup()) < 1 )           /* initiallize monitor */
        {
                fprintf(stderr,"%s: startup failed, monitor error:%d\n",
                        argv[0],status);
                exit(1);
        }
        u_boolvec[0] |= 1;              /* enable AcctHeader */
        u_boolvec[0] |= 1<<1;           /* enable UserXamplSensor */
        strcpy((char *)(&charcvt[0]),"demo run");
        AcctHeader("acct","MONITOR",charcvt);

        /*
         * put in data
         */
#ifndef NOKERN
#ifdef ALL
        enablesensors(1);               /* "NameStart"      */
        enablesensors(2);               /* "NextComponent"  */
        enablesensors(3);               /* "INodeCreate"    */
        /* there is no 4 */
        enablesensors(5);               /* "OpenSuccessful" */
        /* enablesensors(6);            "FileClose"      */
        enablesensors(7);               /* "INodeDelete"    */
#endif
        enablesensors(8);               /* "ReadSensor"     */
#ifdef ALL
        enablesensors(9);               /* "WriteSensor"    */
#endif
#endif


        /*
         * This is the heart of the accountant.
         * in real accountant a signal ends the loop
         * and sleep time is dynamically sized
         * according to usage.
         */

        strcpy((char *)(&charcvt[0]),"acct");
        strcpy((char *)(&charcvt2[0]),"loop");
        for (runminutes = 0; runminutes < runduration; runminutes++)
        {
        /*
```

```
        * break up minute into portions to prevent buffer overflow
        */
#ifdef NOKERN
                for (i = 0; i < 60; i++)
                {
                        UserXamplSensor(1, charcvt, runminutes, charcvt2, rund
uration);
                        UserXamplSensor(1, charcvt, runminutes, charcvt2, rund
uration);
                        UserXamplSensor(1, charcvt, runminutes, charcvt2, rund
uration);
                        sleep(1);               /* some time to get some data
*/
                }
                getevents();            /* puts in global buffer */
#else
                UserXamplSensor(1, charcvt, runminutes, charcvt2, runduration)
;
                for (i =0; i< 20; i++)
                {
                        sleep(3);               /* some time to get some data
*/
                        getevents();            /* puts in global buffer */
                }
#endif
        }
        shutdown();                     /* all done; clean up */
}

/*
 * Routines that call sysl_monitor
 */
/*
 *-----------------------------------------------------------------------
 * startup
 *      - initializes monitor, puts size of event buffer to stdout
 *-----------------------------------------------------------------------
 */
startup()
{
        struct mon_cmd command;
        int     i;

        command.type = MONOP_INIT;
        command.length  = (sizeof(struct mon_cmd)+1)/2;
        i = syscall(SYSLOCAL,SYSL_MONITOR, (unsigned char *)&command);
        fprintf(stderr,"miniacct: init -- %d\n",i);
        return(i);
}
/*
 *-----------------------------------------------------------------------
 * enablesensors
 *      - enables the sensor with eventnumber eventnumber
 *      - prints status of system call
 *      - caller must be accountant
 *-----------------------------------------------------------------------
 */
enablesensors(eventnumber)
int     eventnumber;
{
        mon_putreq      preq;
        int     i;

        preq.cmd.type = MONOP_PUTREQ;
        preq.cmd.length  = (sizeof(mon_putreq)+1)/2;
        preq.req.targetpid   = 0;
        preq.req.eventnumber = eventnumber;
        preq.req.enablevalue = 1;
        i = syscall(SYSLOCAL,SYSL_MONITOR, (unsigned char *)&preq);
        fprintf(stderr,"miniacct: enable -- %d\n",i);
}
/*
 *-----------------------------------------------------------------------
 * shutdown
 *      - closes down monitor
 *      - caller must be accountant
 *-----------------------------------------------------------------------
 */
shutdown()
{
        struct mon_cmd command;
        int     i;

        command.type = MONOP_SHUTDOWN;
        command.length  = sizeof(struct mon_cmd);
        i = syscall(SYSLOCAL,SYSL_MONITOR, (unsigned char *)&command);
        fprintf(stderr,"miniacct: shutdown -- %d\n",i);
}
/*
 *-----------------------------------------------------------------------
 * getevents
 *      - writes out events from monitor's buffer
 *      - prints amount of buffer used
 *-----------------------------------------------------------------------
 */
getevents()
{
        int             i;
        mon_getevent    gevt;
        mon_putevent    *pevt = (mon_putevent *)buffer;
        short           *pos  = buffer;
```

```
        gevt.cmd.type = MONOP_GETEVENTS;
        gevt.cmd.length  = (sizeof(mon_getevent) +1) / 2;
        gevt.req_length = sizeof(buffer)/2;
        gevt.acct_buf_ptr = buffer;
        i = syscall(SYSLOCAL,SYSL_MONITOR, (unsigned char *)&gevt);
#ifdef DEBUG
        fprintf(stderr,"miniacct: getevents -- %d\n",i);
        for ( ; i > 0; (i -= pevt->cmd.length, pos += pevt->cmd.length))
        {
                short   *f;

                pevt = (mon_putevent *) pos;
                printf("command:%d length:%d ",
                        pevt->cmd.type,
                        pevt->cmd.length);
                printf("evt:%d perf:%d obj:%04lx init:%d ",
                        pevt->eventnumber,
                        pevt->performer,
                        pevt->object,
                        pevt->initiator);
                /*
                 * print out fields
                 */
                printf("fields:");
                for (   f = (short *)(pevt->fields);
                        f < ( pos + pevt->cmd.length );
                        f++
                    )
                        printf(" %02x", *f);
                printf("\n");
        }
#else
        write(1, (unsigned char *) buffer, i*2);
#endif
}
```

AGGROP

README

```
Aggrop is divided into the following files:
        aggrop.h                global types and defines
        args.c                  command line processor
        find.c                  handles storage of aggregates
        main.c                  man routines
```

makefile

```
CFILES = args.c find.c main.c
OFILES = args.o find.o main.o
LIB = ../lib
LIBFILES = ../lib/streamio.o ../lib/tuple.o ../lib/schema_idl.o \
        ../lib/readrecord.o ../lib/writerecord.o
CFLAGS = -g -I$(LIB)

aggrop: $(OFILES)
        cc -o aggrop $(CFLAGS) $(OFILES) $(LIBFILES) \
                /usr/softlab/lib/libidl.a

depend:
        -rm -f /tmp/dep
        egrep "^#include" $(CFILES) | grep -v '<' | sed -e "/<.*>/d" \
                -e 's/:[        ]*\#include[     ]*\"\(.*\)\".*$$/:       \1/' \
                -e "s/\.c/.o/" > /tmp/dep
        -rm -f /tmp/sedfile
        touch /tmp/sedfile
        -for i in `awk "{print $$2}" /tmp/dep` ; \
        do for l in $(LIB) ; \
                do if [ -f $$l/$$i ] ; \
                        then    echo  >> /tmp/sedfile "s,$$i,$$l/$$i," ; \
                                break ; \
                        fi ; \
                done ; \
        done
        sed -f /tmp/sedfile /tmp/dep > /tmp/dep2
        sed -e '/^\# Dependencies/,$$ d' makefile > /tmp/makefile
        echo "# Dependencies DON'T REMOVE THIS LINE" \
                | cat - /tmp/dep2 >> /tmp/makefile
        mv makefile makefile.old
        cp /tmp/makefile makefile
        -rm -f /tmp/dep /tmp/dep2 /tmp/sedfile /tmp/makefile

# Dependencies DON'T REMOVE THIS LINE
args.o: aggrop.h
find.o: aggrop.h
main.o: ../lib/schema_idl.h
main.o: aggrop.h
main.o: ../lib/tuple.h
main.o: ../lib/streamio.h
```

aggrop.h

```
/* aggrop.h - global includes, defines, and types for aggrop */

/*      have to hide this from other files
static char rcsid[] = "$Header$";
*/

/*
 * Defines for types
 */

#define NUMERIC 0
#define CHARACTER 1
#define MAXINT  0x7fffffff

/*
 * Defines for options
 */

#define SUM 1
#define AVG 2
#define CT  4
#define MIN 8
#define MAX 16
```

```
union val {                             /* union for domain value */
        char    *cval;
        long    nval;
        };

struct agg_key {                                /* struct for lookup key */
        int     type;
        union val       u;
        };
struct bucket {                         /* buckets for operations */
        long    sum;
        long    ct;
        long    min;
        long    max;
        };
typedef int bool;
```

---

main.c

```
/* main.c - main getdomainval */

/* ************************************************************************ *\
*                                                                            *
*   Title:      aggrop                                                       *
*   Filename:   main.c                                                       *
*   Last Edit: "Mon Dec  9 19:35:12 EST 1985"                                *
*   Author:     Stephen E. Duncan <duncans@unc>                              *
*               Department of Computer Science                               *
*               University of North Carolina                                 *
*               Chapel Hill, NC  27514                                       *
*                                                                            *
*   Copyright (C) The University of North Carolina, 1985                     *
*                                                                            *
*   All rights reserved. No part of this software may be sold or             *
*   distributed in any form or by any means without the prior written        *
*   permission of the SoftLab Software Distribution Coordinator.             *
*                                                                            *
*   Report problems to  softlab@unc (csnet) or                               *
*                       softlab!unc@CSNET-RELAY (ARPAnet)                    *
*   Direct all inquiries to the SoftLab Software Distribution                *
*       Coordinator, at the above addresses.                                 *
*                                                                            *
*   Function: driving routines of aggrop                                     *
*                                                                            *
*           visible: main, getdomainval                                      *
*                                                                            *
\* ************************************************************************ */

#ifndef lint
static char rcsid[] = "$Header:$";
#endif lint

/* ************************************************************************ *\
*   Revision Log:                                                            *
*       $Log:$                                                               *
*                                                                            *
*   Edit Log:                                                                *
*       Oct 19 1985 (duncans) Created                                        *
*       Dec 9 1985 (duncans) Finished standalone testing.                    *
*                                                                            *
\* ************************************************************************ */

#include <stdio.h>
#include <monitor/monops.h>
#include <monitor/montypes.h>
#include "schema_idl.h"
#include "aggrop.h"
#include "tuple.h"
#include "streamio.h"

/*
 *----------------------------------------------------------------------------
 * main -
 *      Processes command line, and
 *      for each tuple in the input stream updates an entry
 *      via find() for each aggregate operation specified
 *      in the command line.
 *      At EOF, uses iterate() to retrieve the values which
 *      are then printed.
 *----------------------------------------------------------------------------
 */

main(argc,argv)
int     argc;
char    **argv;
{
        Mstream *sp;                    /* Monitor stream               */
        tuple   tup;                    /* holds incoming tuple         */
        char    *rel_name,              /* relation to check            */
                *part_name,             /* domain to partition on       */
                *arg_name;              /* domain to perform op         */
        struct agg_key
                *part_key,              /* key for storing value        */
                *tupval;                /* value of argument domain     */
```

```
    struct agg_key *getdomainval();
    register struct bucket *val;    /* stored values                */
    struct bucket *find(),
            *iterate();             /* functions for data storage   */
    void    init_iterate();
    int     flags,                  /* OPTIONS - SUM CT AVG MIN MAX */
            status,                 /* status of calls              */
            rel_id,                 /* numeric id of relation       */
            all_relations=0,        /* boolean for restricting rel. */
            err_recd_ct=0,          /* count of error recds in stream */
            missing_dom_ct=0;       /* count of recds missing domains */

/*
 * Get command line arguments.
 */
    args(argc, argv, &flags, &rel_name, &part_name, &arg_name);

    if ( (sp = str_open(stdin)) <= NULL )   /* open a stream */
    {
            fprintf(stderr,"%s.main : can\'t open stream\n");
            exit(1);
    }

/*
 * Check the arguments with the schema
 */
    if ( str_schemaread(sp) <= NULL )       /* need the schema now */
    {
            fprintf(stderr,"%s.main : can't read stream\n");
            exit(1);
    }
    if ( strcmp(rel_name,"-") == 0 )        /* can't check domains */
            all_relations = TRUE;
    else                                    /* can check domains */
    {
            relation        rel_p;          /* pointer for relation */
            attribute       par_p, arg_p;   /* ptrs for domains */

            if ( (rel_p = getrelationbyname(sp->schema,rel_name)) == NULL
)
            {
                    fprintf(stderr, "Invalid relation %s for schema\n",
                            rel_name);
                    exit(1);
            }
            rel_id = rel_p->rel_sensor_id;  /* for faster compares later *
/

            if ( (par_p = getdomainbyname(rel_p,part_name))
                == NULL
                || (arg_p = getdomainbyname(rel_p,arg_name))
                == NULL )
            {
                fprintf(stderr,
                    "%s.main: invalid domains for relation %s\n",
                    argv[0],rel_name);
                exit(1);
            }
    } /* else */


/*
 * Accumulate the data
 */
    while ( (status = str_read(sp,&tup)) > 0 )
    {
        if ( ((struct mon_cmd *)tup.record)->type == MONOP_OFLOW )
        {
            err_recd_ct++;
        }
        else if ( all_relations || rel_id ==
            tup.relation->rel_sensor_id )
        {
            part_key = getdomainval(&tup,part_name);
            tupval = getdomainval(&tup,arg_name);
            if (part_key == NULL || tupval == NULL)
                if (all_relations)          /* this is allowed */
                {
                    missing_dom_ct++;       /* keep track of how many */
                    continue;               /* bypass rest of loop */
                }
                else
                {
                    fprintf(stderr,
                      "%s.main : %s missing partition or argument\n",
                      argv[0], tup.relation->rel_name);
                    exit(1);
                } /* else */
            if (tupval->type != NUMERIC)
            {
                fprintf(stderr,
                    "%s.main : %s has invalid type\n",
                    argv[0], arg_name);
                exit(1);
            } /* if */
            val = find(part_key);
            val->sum += (flags&(SUM|AVG)) ? tupval->u.nval : 0;
            val->ct  += (flags&(CT|AVG)) ? 1 : 0;
            val->min  = (flags&MIN && tupval->u.nval < val->min) ?
                    tupval->u.nval : val->min;
            val->max  = (flags&MAX && tupval->u.nval > val->max) ?
                    tupval->u.nval : val->max;
```

```
        } /* else if */
    } /* while */
    if ( status < 0 )                   /* an io error occurred */
    {
            fprintf(stderr,"%s.main : stream read error\n",argv[0]);
            exit(1);
    }

/*
 * Print out error counts, if any.
 */
    if ( err_recd_ct > 0 )
            fprintf(stderr,"Error records in stream: %d\n", err_recd_ct);
    if ( all_relations && missing_dom_ct > 0 )
            fprintf(stderr,"Records in stream missing domains: %d\n",
                    missing_dom_ct);

/*
 * Print out the results
 */
    init_iterate();                     /* initialize for iterations */
    while ( (val = iterate()) != NULL )
    {
            /*
             * Only print requested ops.
             * Could use sprintf if this is too slow
             */

            if (flags&CT)
                    printf("%d",val->ct);
            if (flags&SUM)
                    printf("\t%d",val->sum);
            if (flags&AVG)
            {
                if (val->ct)
                    printf("\t%f", (float)val->sum / (float)val->ct);
                else
                    printf("\t0");
            }
            if (flags&MIN)
                    printf("\t%d",val->min);
            if (flags&MAX)
                    printf("\t%d",val->max);
            printf("\n");
    } /* while */
    exit(0);
} /* main */

/*
 *----------------------------------------------------------------------
 * getdomainval -
 *      search the tuple for the domain and return its value
 *----------------------------------------------------------------------
 */
struct agg_key *
getdomainval(tp,domname)
tuple   *tp;
char    *domname;
{
        attribute       ap;             /* domain description in schema */
        int             position;       /* position of domain in record */
        struct agg_key  *part_key =     /* new key to return */
                (struct agg_key *)malloc(sizeof(struct agg_key));

        if ( (ap = getdomainbyname(tp->relation,domname)) == NULL )
                return(NULL);           /* doesn't have relation */
        position =  ap->attr_pos;       /* displacement in shorts of domain */
        if (position < 0) position *= -1;
        switch (typeof(ap->attr_type))
        {
            case Ktype_string:          /* just point to it, don't copy it */
                part_key->type = CHARACTER;
                part_key->u.cval = (char *)&(tp->record[position]);
                break;
            case Ktype_boolean:         /* treat these by length */
            case Ktype_integer:
            case Ktype_rational:
                part_key->type = NUMERIC;
                switch(ap->attr_length)
                {
                    case 1:
                        part_key->u.nval = (tp->record[position]);
                        break;
                    case 2:             /* take two bytes */
                        part_key->u.nval = *(short *)(&tp->record[position]);
                        break;
                    default:            /* assume we take four bytes */
                        part_key->u.nval = *(long *)(&tp->record[position]);
                        break;
                } /* switch */
                break;
            default:
                return(NULL);
        } /* switch */
        return(part_key);
} /* getdomainval */
```

args.c

```
/* args.c - arg */

/* ************************************************************************ *\
*                                                                            *
*   Title:      aggrop                                                       *
*   Filename:   args.c                                                       *
*   Last Edit: "Mon Dec  9 19:38:40 EST 1985"                                *
*   Author:     Stephen E. Duncan <duncans@unc>                              *
*               Department of Computer Science                               *
*               University of North Carolina                                 *
*               Chapel Hill, NC  27514                                       *
*                                                                            *
*   Copyright (C) The University of North Carolina, 1985                     *
*                                                                            *
*   All rights reserved. No part of this software may be sold or             *
*   distributed in any form or by any means without the prior written        *
*   permission of the SoftLab Software Distribution Coordinator.             *
*                                                                            *
*   Report problems to  softlab@unc (csnet) or                               *
*                       softlab!unc@CSNET-RELAY (ARPAnet)                    *
*   Direct all inquiries to the SoftLab Software Distribution                *
*       Coordinator, at the above addresses.                                 *
*                                                                            *
*   Function: process arguments for aggrop                                   *
*                                                                            *
*           visible: args                                                    *
*                                                                            *
\* ************************************************************************ */

#ifndef lint
static char rcsid[] = "$Header:$";
#endif lint

/* ************************************************************************ *\
*   Revision Log:                                                            *
*       $Log:$                                                               *
*                                                                            *
*   Edit Log:                                                                *
*       Oct 19 1985 (duncans) Created                                        *
*       Dec 9 1985 (duncans) Finished standalone testing.                    *
*                                                                            *
\* ************************************************************************ */

#include <stdio.h>
#include "aggrop.h"

/*
 *------------------------------------------------------------------------
 * arg -
 *      Set flags and grab arguments from command line.
 *      Exits with 1 upon error.
 *------------------------------------------------------------------------
 */

#define USAGE "Usage: aggrop -ascnx relation partition argument\n"

args(argc, argv, options, relation, partition, argument)
int     argc,                   /* Number of arguments */
        *options;               /* Boolean array of operations */
char    **argv,                 /* Array of arguments */
        **relation,             /* Relation to apply operations */
        **partition,            /* Domain to partition aggregates */
        **argument;             /* Operand domain of aggregate operation */
{
        int     i,j,            /* Array subscripts */
                op_test = 0;    /* Count of operands */

        *options = 0;           /* Clear options */
        argv++;                 /* bypass program name */
        argc--;
        for ( i = 0; i < argc; i++, argv++ )
        {
                /*
                 * A null argv[1] means the univeral relation.
                 */
                if ( *(*argv+0) == '-' && *(*argv+1) )  /* test for option */
                {
                /*
                 * Options can be individually specified or
                 * all lumped together behind a single '-'
                 */
                        for ( j = 1; *(*argv + j) ; j++ )
                        {
                                switch (*(*argv + j))
                                {
                                case 'a':
                                        *options |= AVG;
                                        break;
                                case 's':
                                        *options |= SUM;
                                        break;
                                case 'c':
                                        *options |= CT;
                                        break;
                                case 'n':
```

```
                        *options |= MIN;
                        break;
                case 'x':
                        *options |= MAX;
                        break;
                default:
                        fprintf(stderr,
                                "Unknown option: %c\n",
                                *(*argv + j));
                        exit(1);
                } /* switch */
        } /* for j */
} /* if */
else                    /* select operand */
{
/*
 * This section sets the operands depending on
 * how many have been seen.  Note that storage
 * must be provided for each.  The default part
 * lets us quit upon an error
 */
        switch(op_test)
        {
        case 0:
                *relation = (char *)malloc(strlen(*argv)+1);
                strcpy(*relation, *argv);
                op_test++;
                break;
        case 1:
                *partition = (char *)malloc(strlen(*argv)+1);
                strcpy(*partition, *argv);
                op_test++;
                break;
        case 2:
                *argument = (char *)malloc(strlen(*argv)+1);
                strcpy(*argument, *argv);
                op_test++;
                break;
        default:        /* Too many operands! */
                fprintf(stderr, USAGE);
                exit(1);
        } /* switch */
} /* else */
} /* for i */

/*
 * Make sure that we have what is needed.
 */

if (op_test != 3)               /* Need 3 operands */
{
        fprintf(stderr, USAGE);
        exit(1);
}
if (*options == 0) *options = CT;       /* Default action */
} /* arg */
```

---

find.c

```
/* find.c - find init_iterate iterate */

/* ************************************************************************ *\
*                                                                            *
*   Title:      aggrop                                                       *
*   Filename:   find.c                                                       *
*   Last Edit: "Mon Dec  9 19:42:08 EST 1985"                                *
*   Author:     Stephen E. Duncan <duncans@unc>                              *
*               Department of Computer Science                               *
*               University of North Carolina                                 *
*               Chapel Hill, NC  27514                                       *
*                                                                            *
*   Copyright (C) The University of North Carolina, 1985                     *
*                                                                            *
*   All rights reserved. No part of this software may be sold or             *
*   distributed in any form or by any means without the prior written        *
*   permission of the SoftLab Software Distribution Coordinator.            *
*                                                                            *
*   Report problems to  softlab@unc (csnet) or                               *
*                       softlab!unc@CSNET-RELAY (ARPAnet)                    *
*   Direct all inquiries to the SoftLab Software Distribution                *
*       Coordinator, at the above addresses.                                 *
*                                                                            *
*   Function: handle structures for aggregates for aggrop                    *
*                                                                            *
*           visible: find init_iterate iterate                               *
*           hidden: rfind                                                    *
*                                                                            *
\* ************************************************************************ */

#ifndef lint
static char rcsid[] = "$Header:$";
#endif lint

/* ************************************************************************ *\
```

```
*   Revision Log:                                                       *
*       $Log:$                                                          *
*                                                                       *
*   Edit Log:                                                           *
*       Oct 19 1985 (duncans) Created                                   *
*       Dec 9 1985 (duncans) Finished standalone testing.               *
*                                                                       *
\* ********************************************************************* */

#include <stdio.h>
#include "aggrop.h"

/*
        This module implements an abstract structure containing a
        set of <key,value> pairs.
        The only operators are find and iterate.
        Find takes a key and searches the private structure for it,
        allocating a new one if not found.
 */

/*
 *----------------------------------------------------------------------
 * agglist -
 *      private data structure containing a key and a value.
 *----------------------------------------------------------------------
 */

struct bucket *find();
static struct bucket *rfind();

static struct entry {
        struct agg_key  key;            /* key for this entry */
        struct bucket   bucket;         /* values */
        struct entry    *next;          /* linked list */
        } *agglist = NULL,              /* list of values */
        *rlist = NULL;                  /* Iterate() position in agglist */


/*
 *----------------------------------------------------------------------
 * find -
 *      search for the key in the data structure agglist,
 *      if one doesn't exist, add it.
 *      return a pointer to the bucket in the entry
 *----------------------------------------------------------------------
 */

struct bucket *
find(key)
struct agg_key  *key;
{
```



```
        /*
         * Only check is for potential mess up of key.
         */
        if (agglist != NULL && key->type != agglist->key.type)
        {
                fprintf(stderr,"find: Bad key type\n");
                exit(1);
        }
        /*
         * Actually starts up recursive lookup with global agglist
         */
        return (rfind(&agglist,key));
}
/*
 *----------------------------------------------------------------------
 * rfind -
 *      Does real work, traverses agglist looking for match on key
 *----------------------------------------------------------------------
 */
#define NEWENTRY (struct entry *)malloc(sizeof(struct entry))

static struct bucket *
rfind(agglist,key)
struct entry    **agglist;              /* aggregate bucket list, never NULL *
/
struct agg_key  *key;                   /* lookup key */
{
        /*
         * This searches a sorted linked list for a key
         * and returns a pointer to the buckets for that key.
         * The key may be numeric or character, and the entry
         * may have to be created.
         */

        int     sw;                             /* 3 way decision variable */
        register struct entry *ep;              /* pointer to list entry */

        if ( *agglist != NULL)                  /* Can still check */
        {
                sw = (key->type == NUMERIC) ?   /* put in form of strcmp */
                        key->u.nval - (*agglist)->key.u.nval
                        : strcmp(key->u.cval,(*agglist)->key.u.cval);
                if ( sw == 0 )                  /* found it */
                        return(&((*agglist)->bucket));
                if ( sw > 0 )                   /* keep looking */
                        return(rfind((*agglist)->next,key));
                /*
                 * If sw < 0, we should insert
                 * the entry here, so just fall through.
                 */
        }
```

```
        /*
         * Insert a new entry here
         * This was reached by either a null list
         * or the value is prior to the current list value.
         */
        if ( (ep = NEWENTRY) == NULL )  /* create a new entry */
        {
                fprintf(stderr, "rfind: malloc failed\n");
                exit(1);
        }
        if (key->type == NUMERIC)       /* initialize key */
        {
                ep->key.type = NUMERIC;
                ep->key.u.nval = key->u.nval;
        }
        else                            /* character key */
        {
                ep->key.type = CHARACTER;
                ep->key.u.cval = (char *)malloc(strlen(key->u.cval) +1);
                strcpy(ep->key.u.cval, key->u.cval);
        }
        ep->bucket.sum = 0;             /* initialize buckets */
        ep->bucket.ct = 0;
        ep->bucket.min = MAXINT;
        ep->bucket.max = 0;
        /*
         * Insert the new entry into the list
         */
        ep->next = *agglist;            /* set up tail of list */
        *agglist  = ep;                 /* link into list */
        return(&((*agglist)->bucket));
} /* rfind */

/*
 *----------------------------------------------------------------------
 * init_iterate -
 *      Resets rlist to agglist, so iterate() can start from
 *      beginning again.
 *----------------------------------------------------------------------
 */
void
init_iterate()
{
        rlist = agglist;
} /* init_iterate */

/*
 *----------------------------------------------------------------------
 * iterate -
 *      Each time the routine is called, return the next value in
 *      agglist.  Iterate is reinitialized by init_iterate.
 *      NULL is returned at end of list.
 *----------------------------------------------------------------------
 */
struct bucket *
iterate()
{
        struct bucket   *bucket;
        if (rlist)                      /* Check for end of list */
        {
                bucket = &(rlist->bucket);
                rlist  = rlist->next;   /* Set up for next call */
                return(bucket);
        }
        else
                return(NULL);   /* At end of list */
} /* iterate */
```

APPLYOP

README

```
New version of applyop -- uses pty's.

First, do schema handling, same as old one.

Parts of system:
        handle new schema
        get descriptors for talking to co-routine
        fork co-routine with pty as stdin,stdout,stderr
        loop:
                read tuple
                if selected,
                        prject domains to string
                        write string to pty
                        read from pty
                        append to tuple
                write tuple
```

makefile

```
CFILES = main.c args.c modify_schema.c pty.c co_routine.c apply.c tty.c
OFILES = main.o args.o modify_schema.o pty.o co_routine.o apply.o tty.o
LIB = ../lib /usr/include/monitor
LIBFILES = ../lib/streamio.o ../lib/tuple.o ../lib/schema_idl.o \
        ../lib/readrecord.o ../lib/writerecord.o
CFLAGS = -DDEBUG -DPTYS -g -I../lib -I/usr/include/monitor

applyop: $(OFILES) $(LIBFILES)
        cc -o applyop $(CFLAGS) $(OFILES) $(LIBFILES) \
                /usr/softlab/lib/libidl.a

lint:
        lint -DDEBUG -I../lib -I/usr/include/monitor $(CFILES)

depend:
        egrep "^#include" $(CFILES) | grep -v '<' | sed -e "/<.*>/d" \
        -e 's/:[       ]*\#include[    ]*\"\(.*\)\".*$$/:       \1/' \
        -e "s/\.c/.o/" > /tmp/dep
        -for i in 'awk '{print $$2}' /tmp/dep | sort | uniq' ; \
        do for l in . $(LIB) ; \
                do if [ -f $$l/$$i ] ; \
                        then    echo  "s,$$i,$$l/$$i," ; \
                                break ; \
                        fi ; \
                done ; \
        done > /tmp/sedfile
        sed -f /tmp/sedfile /tmp/dep > /tmp/dep2
        sed -e '/^\# Dependencies/,$$ d' makefile > /tmp/makefile
        echo "# Dependencies DON'T REMOVE THIS LINE" \
                | cat - /tmp/dep2 >> /tmp/makefile
        mv makefile makefile.old
        cp /tmp/makefile makefile
        -rm -f /tmp/dep /tmp/dep2 /tmp/sedfile /tmp/makefile

# Dependencies DON'T REMOVE THIS LINE
main.o: ../lib/schema_idl.h
main.o: ../lib/tuple.h
main.o: ../lib/streamio.h
args.o: ../lib/schema_idl.h
args.o: ../lib/tuple.h
modify_schema.o:        /usr/include/monitor/montypes.h
modify_schema.o:        ../lib/schema_idl.h
modify_schema.o:        ../lib/tuple.h
co_routine.o:   ../lib/schema_idl.h
co_routine.o:   ../lib/tuple.h
co_routine.o:   ./tty.h
apply.o:        /usr/include/monitor/montypes.h
apply.o:        ../lib/schema_idl.h
apply.o:        ../lib/tuple.h
tty.o:  ./tty.h
```

main.c

```
/* main.c -- main routine of applyop */

#include <stdio.h>
```

```
#include "schema_idl.h"
#include "tuple.h"
#include "streamio.h"


database schemain, schemaout, modify_schema();
char *Toolname;

main(argc,argv)
int     argc;
char    **argv;
{
        char         **co_argv;             /* arg vector to co_routine */
        tuple        tup;                   /* current tuple in stream  */
        SEQrelation  proj_seq_rp;           /* affected relations       */
        SEQattribute app_seq_ap; /* affected relations  */
        Mstream      *sp_in, *sp_out;

        /*
         * Get schema from stdin
         */

        sp_in = str_open(stdin);
        schemain = str_schemaread(sp_in);
        Toolname = *argv;

        /*
         * Process command line args to set up co_argv for coroutine,
         * set up target structure for each target relation, set up
         * new schema.
         */

        args(argc, argv, &co_argv, &proj_seq_rp, &app_seq_ap);
        schemaout = modify_schema(schemain, proj_seq_rp, app_seq_ap);
                                                /* new output schema */
        sp_out = str_open(stdout);
        sp_out->schema = schemaout;


        /*
         * Set up communication and execution of co_routine
         */

        start_coroutine(co_argv);

        /*
         * Main processing loop
         * apply the coroutine to tuples with relations in proj_seq_rp
         */

        while ( str_read(sp_in,&tup) > 0 )
        {
            int         rel_sensor_id = tup.relation->rel_sensor_id;
            SEQrelation t_seq_rp;               /* loop tmps            */
            relation    rp;

            foreachinSEQrelation(proj_seq_rp, t_seq_rp, rp)
                if ( rp->rel_sensor_id = rel_sensor_id )
                {
                        apply_coroutine(&tup, rp, app_seq_ap);
                        break;
                }
            str_write(sp_out,&tup);             /* Always write a tuple */
        }

        /*
         * We should check the status of the child process for cleanup
         * handle in co_routine stuff
         */
} /* main */
```

Appendix D

applyop

---

args.c

```
/* process_args.c - process_args usage*/

#include <stdio.h>
#include <ctype.h>
#include <strings.h>
#include "schema_idl.h"
#include "tuple.h"

extern database schemain, schemaout;            /* schemas from main.c */
extern char  *Toolname;                         /* invoked name        */

/*
 *-----------------------------------------------------------------------
 * args -
 *      Set flags and grab arguments from command line.
 *      Exits with 1 upon error.
 *-----------------------------------------------------------------------
 */

args(argc, argv, prog_argv, proj_seq_rp, app_seq_ap)
int          argc;              /* command line arguments               */
```

```
char        **argv,
            ***prog_argv;       /* argument vector to program           */
SEQrelation  *proj_seq_rp;      /* input relations to be projected      */
SEQattribute *app_seq_ap;       /* output domains to be appended        */
{
        int         i,j,        /* Array indices                        */
                    p = 1,      /* index for prog_argv, 0 is prog_name */
                    op_test = 0; /* Count of operands                   */
        char        *tolcase();
        relation    rp;         /* tmp variable                         */
        SEQrelation t_seq_rp;   /* tmp variable                         */
        attribute   ap;         /* attribute holder for domain          */
#ifdef APPRELATION
        SEQattribute app_seq_ap; /* appending attributes                */
#endif APPRELATION

        argv++;                 /* bypass program name                  */
        argc--;

        initializeSEQrelation(*proj_seq_rp);
        initializeSEQattribute(*app_seq_ap);
        *prog_argv = (char **)malloc(argc*sizeof(char *));

        /*
         * Read until we run out of arguments or until the end of
         * the projected domains, signalled by "-"
         */

        for ( i = 0; i < argc && strcmp(*argv,"-") != 0; i++, argv++ )
        {
            /*
             * A null argv[1] means the "univeral relation".
             * All options must come before output domains
             */
            if ( *(*argv+0) == '-' && *(*argv+1) )   /* test for option */
            {
            /*
             * Each instance of "-p" requires a following argument,
             * with or without intervening spaces.
             * Flag type options can be individually specified or
             * all lumped together behind a single '-'.
             * Right now, there is only one option, but just in case.
             */
                for ( j = 1; *(*argv + j) ; j++ )
                {
                    if ( *(*argv + j) == 'p' )
                    {
                        /*
                         * Argument to be passed to prog.  Check to see
                         * if intervening space, and assign to next spot
                         * in prog_argv.  The calling routine has the
                         * responsibility to make sure prog_argv has
                         * enough room. Can't use a switch here since
                         * break wouldn't work.  Additional options will
                         * 'else if' or change break to goto.
                         */
                        if (strlen(*argv) == 2) /* in next argument     */
                            if ( ++i < argc )   /* we still have an arg */
                                (*prog_argv)[p++] = *++argv;
                            else
                                usage();
                        else                    /* follows w/out space  */
                            (*prog_argv)[p++] = *argv + 2;
                        break;                  /* exit loop            */
                    }
                    else
                        usage();
                } /* for j */
            } /* if */
            else                                /* select operand       */
            {
            /*
             * This section sets the operands depending on
             * how many have been seen.
             */
                switch(op_test++)
                {
                case 0:                         /* pathname of program  */
                    *(*prog_argv) = *argv;      /* first element        */
                    break;
                case 1:                         /* relation             */
                    if ( strcmp(*argv,"-") == 0 )
                    {
                        relation new_rp;        /* tmp for making seq.  */
                        /*
                         * All relations, so copy all of them.
                         */
                        foreachinSEQrelation(schemain->relations,
                            t_seq_rp,rp)
                        {
                            /*
                             * A separate relation and attribute seq
                             * is needed for project and append
                             */

                            new_rp = copyrelation(rp);
                            initializeSEQrelation(new_rp->attributes);
                            appendrearSEQrelation(*proj_seq_rp,new_rp);
#ifdef APPRELATION
                            new_rp = copyrelation(rp);
                            initializeSEQrelation(new_rp->attributes);
                            appendrearSEQrelation(*app_seq_rp,new_rp);
```

```
#endif APPRELATION
                    }
                }
                else if ( (rp = getrelationbyname(schemain, *argv))
                    == NULL )
                {
                    fprintf(stderr, "%s: Relation not in schema\n",
                            Toolname);
                    exit(1);
                }
                else
                {
                    /*
                     * This relation is the only one in seq
                     */
                    relation new_rp;        /* tmp for making seq.  */

                    new_rp = copyrelation(rp);
                    initializeSEQrelation(new_rp->attributes);
                    appendrearSEQrelation(*proj_seq_rp,new_rp);

#ifdef APPRELATION
                    new_rp = copyrelation(rp);
                    initializeSEQrelation(new_rp->attributes);
                    appendrearSEQrelation(*app_seq_rp,new_rp);
#endif APPRELATION
                }
                break;
            default:                        /* the projected domains  */
                /*
                 * Add the domain to every relation to be projected
                 */
                foreachinSEQrelation(*proj_seq_rp, t_seq_rp, rp)
                {
                    /*
                     * Only add the domain if it is in the relation,
                     * otherwise purge the relation as invalid.
                     * Use same actual attribute so that it is set
                     * when the tuple is read.
                     */
                    relation  old_rp;
                    attribute ap;

                    old_rp = getrelationbysensorid(schemain,
                        rp->rel_sensor_id);
                    if ( (ap = getdomainbyname(old_rp,*argv))
                        != NULL )
                        appendrearSEQattribute(rp->attributes,ap);
                    else
                    {
                        removeSEQrelation(*proj_seq_rp,rp);
```



```
#ifdef APPRELATION
                        removeSEQrelation(*app_seq_rp,rp);
#endif APPRELATION
                    }
                }
            } /* switch */
        } /* else */
    } /* for */


    /*
     * Make sure that we have something to work on.
     */
    if (op_test < 3)                            /* not enough args */
        usage();

    /*
     * Finish up prog_argv
     */
    (*prog_argv)[p] = 0;                        /* terminates list */

    /*
     * Gather up the domains to be appended
     * Note that there may not be any.
     */
    if ( i < argc )                             /* bypass "-"      */
    {
        i++;
        argv++;
    }
    for ( ; i < argc ; i++, argv++ )
    {
        /*
         * Set up the attributes for each domain specified.
         * This comes in the form: "name:type".  The length
         * attribute is determined from the type, while
         * the position attribute is determined later.
         */

        char *name = *argv,                     /* domain's name */
             *d_type;                           /* domain's type */

        if ( (d_type = index(name,':')) == NULL )
            usage();                    /* they forgot the type       */
        else
            *d_type++ = '\0';           /* end name and point to type */
        ap            = Nattribute;     /* allocate a new domain      */
        ap->attr_name = (char *)GetHeap( (d_type - name) );
        strcpy(ap->attr_name,name);

        /*
```

```
            * Set up type and length attributes of domain
            */

           if ( strcmp("charstring",d_type) == 0 )
           {
               (int)ap->attr_type.Vtype_string = Ktype_string;
               ap->attr_length = -2; /* signifies minimum and variable */
           }
           else if ( strcmp("boolean",d_type) == 0 )
           {
               (int)ap->attr_type.Vtype_boolean = Ktype_boolean;
               ap->attr_length = 1;  /* has implications for alignment */
           }
           else if ( strcmp("int",d_type) == 0 )
           {
               (int)(ap->attr_type.Vtype_integer) = Ktype_integer;
               ap->attr_length = sizeof(short);
           }
           else if ( strcmp("double",d_type) == 0 )
           {
               (int)(ap->attr_type.Vtype_integer) = Ktype_integer;
               ap->attr_length = sizeof(long);
           }
           else if ( strcmp("rational",d_type) == 0 )
           {
               (int)(ap->attr_type.Vtype_rational) = Ktype_rational;
               ap->attr_length = sizeof(float);     /* same as a float */
           }
           else
           {
               fprintf(stderr, "%s: Invalid type for output domain\n",
                       Toolname);
               _exit(1);
           }
           appendrearSEQattribute(*app_seq_ap, ap);
       } /* for */
#ifdef APPRELATION
       /*
        * Put appending sequences in each appending relation
        */

       foreachinSEQrelation(*app_seq_rp, t_seq_rp, rp)
           rp->attributes = app_seq_ap;
#endif APPRELATION
} /* process_args */

/*
 *----------------------------------------------------------------------
 * usage -
 *      prints usage error message and exits
 *----------------------------------------------------------------------
 */
usage()
{
       fprintf(stderr, "Usage: %s [-p] prog relation domain [domain] = result
domain:type [resultdomain:type]\n", Toolname);
       exit(1);
} /* usage */

/*
 *----------------------------------------------------------------------
 * tolcase -
 *      return a string transformed to all lower case
 *----------------------------------------------------------------------
 */
static char *
tolcase(string_p)
char    *string_p;
{
       char    *new = (char *)malloc(strlen(string_p)+1);
       char    *new_p = new;

       while ( *string_p )
               *new_p++ = tolower(*string_p++);
       *new_p = '\0';
       return(new);
} /* tolcase */
```

---

modify_schema.c

```
/* modify_schema.c - modify_schema */

#include <stdio.h>
#include "montypes.h"
#include "schema_idl.h"
#include "tuple.h"

/*
 *----------------------------------------------------------------------
 * modify_schema -
 *      create a new schema by appending the attributes in app_seq_ap
 *      to each relation in pro_seq_rp.
 *      Return a copy of the new schema.
 *----------------------------------------------------------------------
```

```
 */

extern  char    *Toolname;              /* invoked name from command line */

database
modify_schema(schema, pro_seq_rp, app_seq_ap)
database     schema;
SEQrelation  pro_seq_rp;
SEQattribute app_seq_ap;
{
        SEQrelation  seq_rp;                    /* loop temporary       */
        relation     this_rp;                   /* loop temporary       */
        database     new_schema;                /* schema to be created */

        /*
         * Copy schema to new_schema and modify new_schema.
         */

        new_schema = Ndatabase;
        new_schema->database_name = schema->database_name;
        initializeSEQrelation(new_schema->relations);
        /*
         * Copy the sequence of relations
         */
        foreachinSEQrelation(schema->relations, seq_rp, this_rp)
        {
            /*
             * If this_rp is in pro_seq_rp, use the new
             * one instead.
             */
            relation    new_rp;                 /* rel. in pro_seq_rp   */
            SEQrelation t_seq_rp;               /* loop tmp.            */
            int         rel_sensor_id = this_rp->rel_sensor_id;

            foreachinSEQrelation(pro_seq_rp, t_seq_rp, new_rp)
            {
                if ( rel_sensor_id == new_rp->rel_sensor_id )
                    break;                      /* success              */
                else
                    new_rp = NULL;              /* to indicate failure  */
            }
            if ( new_rp == NULL )
                appendrearSEQrelation(new_schema->relations,this_rp);
            else
            {
                /*
                 * Must get new relation,
                 * create a new sequence of attributes from the old,
                 * append the attributes in app_seq_ap to it, making
                 * sure that rel_vlensensor is accurate.
                 */
```



```
                SEQattribute t_seq_ap;          /* tmp variables        */
                attribute    ap;
                int          pos;               /* position in record   */

                new_rp = copyrelation(this_rp);
                appendrearSEQrelation(new_schema->relations,new_rp);
                retrievelastSEQattribute(new_rp->attributes,ap);
                if ( !new_rp->rel_vlensensor )
                    pos = ap->attr_pos + ap->attr_length;
                foreachinSEQattribute(app_seq_ap, t_seq_ap, ap)
                {
                    appendrearSEQattribute(new_rp->attributes,ap);
                    if (new_rp->rel_vlensensor)
                        ap->attr_pos = -1;
                    else
                    {
                        ap->attr_pos = pos;
                        pos += ap->attr_length;
                        if ( (typeof(ap->attr_type)) == Ktype_string )
                            new_rp->rel_vlensensor = TRUE;
                                            /* make variable length */
                    }
                }

            } /* else - new relation section */
        } /* foreach - setting up SEQrelation in new_schema */

        return(new_schema);

} /* modify_schema */
```


---

tty.h

```
/* tty.h */

/* Author: J. Menges, UNC
 * Modified:
 *      S. Duncan, UNC, 3/18/86: added comments
 */
#include <sgtty.h>
#include <sys/ioctl.h>

typedef struct {
```

```
    char        *name;
    int         fd;
} PTYPORT;

typedef struct {
    PTYPORT     master;
    PTYPORT     slave;
} PTY;

typedef struct {
    struct sgttyb sgttyb;
    struct tchars tchars;
    int ldisc;
    struct ltchars ltchars;
    int lmask;
} TTYATTR;
```

---

co_routine.c

```
/* co_routine.c - start_coroutine, channel_write, channel_read,
 *                channel_error, channel_eof
 */

#include <stdio.h>
#include <signal.h>
#include "schema_idl.h"
#include "tuple.h"
#ifdef PTYS
#include "tty.h"
#endif PTYS

/*
 * These routines are used to start and communicate with a co-routine.
 * Since the implementation is likely to be system dependent, it is
 * isolated in this module.  The main routine doesn't know how the
 * routine is accessed, only that it is.
 * This particular implementation uses 4.2 BSD ptys to run the co-routine.
 * The requirement for any routine is that it will not buffer I/O on
 * more than a line basis.  The co-routine must be able to produce a line
 * for every line given it before processing the next input line.
 */

/*
 * Global communication structures:
 */

#define RTNBUFSIZE 1024                 /* size of IO buffer         */
#define LINEMAX 512                     /* permitted length of line  */
#define TO 1
#define FROM 0
static FILE *channel[2];                /* comm channel to coroutine */
static char buffer[RTNBUFSIZE];         /* IO buffer for result      */
extern char *Toolname;                  /* name of tool              */

/*
 *----------------------------------------------------------------------
 * start_coroutine -
 *      fork and exec a coroutine.
 *      After the fork, dup stdin and stdout to be the ends of the pty.
 *----------------------------------------------------------------------
 */
void
start_coroutine(argv)
char    **argv;
{
#ifdef PTYS
        int     child;                  /* hold the pid of the child */
        PTY     *pty;                   /* pseudo terminal           */
        void    (*sigchld_manage)();    /* handles dead child        */
        PTY     *getpty();

        /*
         * Get a pseudo terminal to run the coroutine on.
         */

        if ( (pty = getpty()) == NULL )
            _exit(1);
        if (ttycopyattr(0, pty->slave.fd) < 0)
            _exit(1);
        if (ttysetecho(pty->slave.fd) < 0)
            _exit(1);
        if (ttysetcbreak(pty->slave.fd) < 0)
            _exit(1);

        if ( (child=ptyexecv(*argv, argv, pty, 0)) < 0 )
        {
            /*
             * Error, abort the process.
             */
            fprintf(stderr,"%s: start_coroutine - fork failed", Toolname);
            _exit(1);
        };
        /*
```

```
         * Parent - do other half of ptys
         */
        if ((channel[0] = fdopen(pty->slave.fd, "w")) == NULL)
            _exit(1);
        if ((channel[1] = fdopen(pty->master.fd, "r")) == NULL)
            _exit(1);
        signal(SIGCHLD, sigchld_manage);
#endif
#ifdef PRWOPEN
        char    argline[LINEMAX];   /* holds argument line to coroutine */
        void    (*sigchld_manage)();/* signal handler for child process */
        /*
         * Convert argv into form that prwopen can handle.
         */
        signal(SIGCHLD,sigchld_manage);
        for ( ; *argv; argv++ )
        {
            strcat(argline,*argv);
            strcat(argline," ");    /* separate the arguments          */
        }
        /*
         * Open channel to coroutine.
         */
        if ( prwopen(argline,channel) < 0 )
        {
            fprintf(stderr,
                "%s:start_coroutine - prwopen failed for %s\n",
                Toolname,argline);
            _exit(1);
        }
#endif
} /* start_coroutine */

/*
 *----------------------------------------------------------------------
 * channel_write -
 *      write a line to the co_routine on the pty
 *      The line must be extracted from the tuple according to the
 *      attributes in rp.
 *----------------------------------------------------------------------
 */
int
channel_write(tp,rp)
tuple    *tp;
relation rp;
{
        int     status;                         /* status of call */
        tuple   tup;                            /* rebuilds tp    */

        /*
         * Create a new tuple using the record from tp and the relation
```



```
         * rp.  Use tupleprint to write this to the channel.
         */
        tup.record = tp->record;            /* make new tuple using rp */
        tup.relation = rp;
        status = tupleprint(channel[TO], &tup,DONTPRINTLABELS);
        if ( status < 0 )
        {
            fprintf(stderr,
                "%s:channel_write - couldn't write to coroutine\n",
                Toolname);
            _exit(1);
        }
        fflush(channel[TO]);                /* make sure it gets there */
        return(status);
} /* channel_write */

/*
 *----------------------------------------------------------------------
 * channel_read -
 *      read a line from the co_routine on the pty.
 *      Convert the line to the format in rp and put it in fields.
 *----------------------------------------------------------------------
 */
int
channel_read(results)
char    **results;
{
        char    *bp = buffer;           /* read into this buffer        */

        fgets(bp, RTNBUFSIZE, channel[FROM]);
        if ( feof(channel[FROM]) )
        {
                *results = NULL;
                return(0);
        }
        if ( ferror(channel[FROM]) )
        {
            fprintf(stderr,"%s:", Toolname);    /* preface to perror    */
            perror("channel_read");
            _exit(1);
        }
        if ( buffer[ strlen(buffer) - 1 ]  != '\n' )
        {
            /*
             * If bp doesn't end in newline, discard data until it does.
             * This has the effect that all of a given tuple's append
             * domains get eaten up.  Put out an error msg too.
             * Calling routine will figure out that it doesn't have
             * enough values.
             * Alternative is to try to get enough space to read line.
             */
```

```
        char        err_bp[RTNBUFSIZE];

        fprintf(stderr,"%s:channel_read - results too large.\n",
            Toolname);
        fprintf(stderr,"%s:data:%d,%s\n",Toolname,strlen(buffer),
            buffer);
        fgets(err_bp, RTNBUFSIZE, channel[FROM]);
        while ( !feof(channel[FROM]) )
        {
            if ( err_bp[ strlen(buffer) - 1 ] == '\n' )
                break;
            fgets(err_bp, RTNBUFSIZE, channel[FROM]);
        }
    }
    *results = buffer;
    return(strlen(buffer));

} /*channel_read */


/*
 * The following are routines to allow an external module to
 * look at the coroutine output channel.
 */


/*
 *---------------------------------------------------------------------
 * channel_error -
 *      check for an IO error on the output channel
 *---------------------------------------------------------------------
 */
int
channel_error()
{
    return(ferror(channel[FROM]));
}

/*
 *---------------------------------------------------------------------
 * channel_eof -
 *      check for eof on the output channel
 *---------------------------------------------------------------------
 */
int
channel_eof()
{
    return(feof(channel[FROM]));
}

/*
```



```
 *---------------------------------------------------------------------
 * sigchld_manage -
 *      handle anything funny from child, like it dying.
 *---------------------------------------------------------------------
 */
static void
sigchld_manage()
{
    fprintf(stderr,"%s:Coroutine status has changed\n",Toolname);
    _exit();
}
```

---

apply.c

```
/* apply.c - apply_coroutine */

#include <stdio.h>
#include <ctype.h>
#include "montypes.h"
#include "schema_idl.h"
#include "tuple.h"

/*
 *---------------------------------------------------------------------
 * apply_coroutine -
 *      pass domains to coroutine
 *      append the output of the coroutine to the tuple
 *---------------------------------------------------------------------
 */

int     done = FALSE;                   /* processing done flag     */
extern  char *Toolname;                 /* name of tool             */
int     max_ints[] = {
        0,
        (0xff-1),
        (0xffff-1),
        (0xffffff-1),
        (0xffffffff-1)
        };                              /* values for given length  */

void
apply_coroutine(tp, rp, app_seq_ap)
tuple       *tp;                        /* affected tuple           */
```

```
relation     rp;                        /* relation that changes     */
SEQattribute app_seq_ap;                /* new domains               */
{
        SEQattribute t_seq_ap;          /* loop variable             */
        attribute    ap;                /* temp. variable            */
        int          i_pos,             /* index of position         */
                     align = TRUE,      /* alignment of domain       */
                     i;                 /* integer result value      */
        float        f;                 /* rational result value     */
        char         *results,          /* string of results domains */
                     *record = tp->record, /* record from tp         */
                     *last,             /* last allowed position     */
                     *pos,              /* position in record        */
                     *res = results;    /* position in results       */
        void         channel_write();



        if ( done ) return;             /* don't append anymore      */
        channel_write(tp,rp);           /* send to the coroutine     */

        /*
         * Only have to read from coroutine if it writes something.
         */

        if ( emptySEQattribute(app_seq_ap) )
            return;                     /* nothing to append         */
        /*
         * We have domains to append.
         */

        if ( channel_read(&results) == 0 )
        {
                done = TRUE;
                return;                 /* all done here             */
        }

#ifdef DEBUG
        fprintf(stderr,"%s",results);
#endif DEBUG
        /*
         * Append each attribute to tuple
         * ensure that there is room in the tuple,
         * ensure that all attributes are present
         */

        ap = getdomainbyname(tp->relation,"cmdlength");
                                        /* find length of tuple */
        i_pos = record[ap->attr_pos] * 2;       /* convert to chars  */
        pos = &(record[i_pos]);                 /* place to append   */
        last = &(record[(sizeof(mon_putevent) - 1)]);
```



```
                                        /* end of tuple              */
        foreachinSEQattribute(app_seq_ap, t_seq_ap, ap)
        {
            /*
             * scan in the value of each domain into
             * the record at pos
             */
            while ( *res && isspace(*res) ) res++;
            if ( ap->attr_length != 1 )
                pos += (pos - record)%2;        /* align field       */
            switch(typeof(ap->attr_type))
            {
            case Ktype_string:
                while ( *res && !isspace(*res) && pos < last )
                {
                    *(pos++) = *(res++);
                    align = !align;
                }

                /*
                 * Have to end string on alignment, since last
                 * is unaligned, we don't have to check it
                 */

                *(pos++) = '\0';
                if ( !align ) *(pos++) = '\0';
                break;

            case Ktype_integer:
                /*
                 * Need a different approach for each length,
                 * with sanity checks for them.
                 */
                if ( pos + ap->attr_length > last )
                {
                    fprintf(stderr,
                        "%s:apply_coroutine - domain %s extends past end of re
cord\n",
                        Toolname, ap->attr_name);
                    _exit(1);                   /* can't continue */
                }
                if ( (i = atoi(res)) > max_ints[ap->attr_length] )
                {
                    /*
                     * Print msg but continue
                     */
                    fprintf(stderr,
                        "%s:apply_coroutine - value truncated, %s:%d\n",
                        Toolname, ap->attr_name, i);
                }
```

```
#ifdef DEBUG
            fprintf(stderr,"%d\n",i);
#endif DEBUG
            while ( isspace(*res) ) res++;
            while ( isdigit(*res) ) res++;
            switch(ap->attr_length)
            {
            case 1:
                /*
                 * Note that this is already aligned
                 */
                *(unsigned char *)pos = i;
                pos++;                          /* update position */
                break;
            case 2:
                *(short *)pos = i;
                pos += 2;                       /* update position */
                break;
            case 4:
                *(long *)pos = i;
                pos += 4;                       /* update position */
                break;
            default:
                /*
                 * Critical error, since length is not defined!
                 */
                fprintf(stderr,
                    "%s:apply_coroutine - unknown length in schema\n",
                    Toolname);
                _exit(1);
            }
            break;

        case Ktype_rational:
            if ( pos + ap->attr_length > last )
            {
                fprintf(stderr,
                    "%s:apply_coroutine - domain %s extends past end of re
cord\n",
                    Toolname, ap->attr_name);
                _exit(1);                       /* can't continue  */
            }
            if ( sscanf(res,"%f",&f) != 1 )
            {
                fprintf(stderr,
                    "%s:apply_coroutine - couldn't read value for %s\n",
                    Toolname, ap->attr_name);
                _exit(1);                       /* can't continue  */
            }
            *(float *)pos = f;
            pos += ap->attr_length;             /* update position */
            break;

        case Ktype_boolean:
            if ( pos + ap->attr_length > last )
            {
                fprintf(stderr,
                    "%s:apply_coroutine - domain %s extends past end of re
cord\n",
                    Toolname, ap->attr_name);
                _exit(1);                       /* can't continue  */
            }
            if ( sscanf(res,"%i",&i) != 1 )
            {
                fprintf(stderr,
                    "%s:apply_coroutine - couldn't read value for %s\n",
                    Toolname, ap->attr_name);
                _exit(1);                       /* can't continue  */
            }
            *(unsigned char *)pos = (i != 0);
            pos++;                              /* update position */
            break;

        default:
            fprintf(stderr,
                "%s:apply_coroutine - invalid type, %s:%d\n",
                Toolname, ap->attr_name, typeof(ap->attr_type));
            _exit(1);                           /* can't continue  */
        } /* switch on typeof */
    } /* foreach */
    return;                                     /* normal return   */
} /* apply_co_routine */
```

---

pty.c

```
/* pty.c - prwopen, prwclose */
/*
 * NAME
 *      prwopen - modified popen(3) to work with pty's instead of
 *                pipes, and also provide both read and write
 *                capabilities to the child process.
 *
 * SYNOPSIS
 *      #include <stdio.h>
```

```
*
*       int prwopen( cmd, streams )
*       char *cmd;
*       FILE streams[2];
*
* AUTHOR (actually, merger!)
*       John Ioannidis, ioannidis@cs.columbia.edu
*       Chris Torek mod made by Steve Duncan
*
* SEE ALSO
*       popen(3), after which this call is modelled.
*
* UNIX SOURCES USED
*       popen(3), script(1)
*
*/
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <signal.h>
#include <sgtty.h>

static  int     pip_pid[ 20 ];

int
prwopen( cmd, streams )
char *cmd;
FILE *streams[];
{
        int pty,                /* file descriptor for pty      */
            pid,                /* pid of cmd                   */
            j;                  /* index into char table        */
        struct stat stb;
        char c;
        static char *line = "/dev/ptyp0";

        for( c = 'p'; ; c++ )           /* Torek Mod: removes condition */
        {
                /*
                 * Check pty's from ptyc0
                 * Invariant: no available pty's before 'line'
                 * Terminates when no more ptys to check
                 */
                line[strlen("/dev/pty")] = c;
                line[strlen("/dev/ptyp")] = '0';
                if( stat( line, &stb ) < 0 )
                        break;          /* loop terminator              */
                for( j = 0; j < 16; j++ )
                {
                        /*
                         * Get particular pty
                         */
                        line[strlen("/dev/ptyp")] = "0123456789abcdef"[j];
                        if( ( pty = open( line, 2 )) > 0 )
                        goto opened;
                }
        }
        return( -1 );                   /* no ptys available            */

opened:
        /*
         * Have found a pty
         */
        switch( pid = fork() )
        {
        case -1:
                return( -1 );

        case 0:
                {
                        int t,                  /* descriptor of old tty  */
                            tty;                /* descriptor of new tty  */
                        struct sgttyb bf;
                        t=open( "/dev/tty", 2 );
                        if( t >= 0 )            /* check for valid        */
                        {
                                ioctl( t, TIOCNOTTY, (char *)0 );
                                close( t );     /* discard parent tty     */
                        }

                        /*
                         * Get equivalent tty for other half of device
                         */

                        line[strlen("/dev/")] = 't';
                        tty = open( line, 2 );
                        close( pty );           /* only used by parent
        */
                        ioctl( tty, TIOCGETP, &bf );
                        bf.sg_flags &= ~ECHO;
                        ioctl( tty, TIOCSETP, &bf );
                        dup2( tty, 0 );         /* reset std{in,out,err}  */
                        dup2( tty, 1 );
                        dup2( tty, 2 );
                        close( tty );           /* all done with this now */

                        execl( "/bin/sh", "sh", "-c", cmd, 0 );
                        _exit(1);
                }
        }
```

```
        pip_pid[pty]=pid;                       /* so we know which one     */

        if( (streams[0] = fdopen( pty, "r" )) == NULL ||
            (streams[1] = fdopen( pty, "w" )) == NULL )
        {
                return( -1 );
        }
        setbuf( streams[0], NULL );             /* get rid of block size    */
        setbuf( streams[1], NULL );             /* get rid of block size    */
        return( pid );
}

prwclose( streams )
FILE *streams[];
{
        register f, r, (*hstat)(), (*istat)(), (*qstat)();
        int status;

        f = fileno(streams[1]);
        fclose(streams[0]);
        fclose(streams[1]);
        istat = signal(SIGINT, SIG_IGN);
        qstat = signal(SIGQUIT, SIG_IGN);
        hstat = signal(SIGHUP, SIG_IGN);
        while((r = wait(&status)) != pip_pid[f] && r != -1)
                ;
        if(r == -1)
                status = -1;
        signal(SIGINT, istat);
        signal(SIGQUIT, qstat);
        signal(SIGHUP, hstat);
        return(status);
}
```

---

tty.c

```
/* tty.c - getptyname, ptyopen, getpty, ttygetattr, ttysetattr,
 * ttycopyattr, ttysetsecho, ttysetcbreak, ptyexecv
 */

/* Author: J. Menges, UNC */
/* Modified:
 *      S. Duncan, UNC, 3/18/86; added comments
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>

#include <stdio.h>

#include "tty.h"

#define PTYTEMPLATE     "/dev/ptyXX"
#define PTYLETTER       'p'
#define PTYDIGITS       "0123456789abcdef"

#define error_pep(string) {perror(string); return(NULL);}
#define error_prp(string)\
        {fprintf(stderr, "%s\n", string); return(NULL);}
#define error_pen(string) {perror(string); return(-1);}
#define error_prn(string) {fprintf(stderr, "%s\n", string); return(-1);}

/*
 *----------------------------------------------------------------------
 * getptyname -
 *      return the name of a pty
 *----------------------------------------------------------------------
 */
static char *
getptyname(i)
{
    static char *template = PTYTEMPLATE;

    template[strlen(PTYTEMPLATE) - 2] = PTYLETTER + i / strlen(PTYDIGITS);
    template[strlen(PTYTEMPLATE) - 1] = PTYDIGITS[i % strlen(PTYDIGITS)];
    return template;
}

/*
 *----------------------------------------------------------------------
 * ptyopen -
 *      return a descriptor to a pty
 *----------------------------------------------------------------------
 */
static int
ptyopen(ptyname)
char *ptyname;
{
    struct stat status;

    return (stat(ptyname, &status) < 0) ? -2 : open(ptyname, 2);
}
```

```
/*
 *----------------------------------------------------------------------
 * getpty -
 *      return a pointer to a PTY struct
 *----------------------------------------------------------------------
 */
PTY *
getpty()
{
    int i;
    char *ptyname;
    PTY *pty;

    pty = (PTY *) malloc(sizeof(PTY));
    if (pty == NULL) error_pep("getpty: malloc");

    for (i = 0; ; i++) {
        ptyname = getptyname(i);
        pty->master.fd = ptyopen(ptyname);
        if (pty->master.fd == -2) error_prp("getpty: no more ptys\n");
        if (pty->master.fd >= 0) {
            pty->master.name = (char *) malloc(strlen(PTYTEMPLATE) + 1);
            if (pty->master.name == NULL) error_pep("getpty: malloc");
            strcpy(pty->master.name, ptyname);

            pty->slave.name = (char *) malloc(strlen(PTYTEMPLATE) + 1);
            if (pty->slave.name == NULL) error_pep("getpty: malloc");
            strcpy(pty->slave.name, ptyname);
            pty->slave.name[strlen(PTYTEMPLATE) - 5] = 't';
            pty->slave.fd = open(pty->slave.name, 2);
            if (pty->slave.fd < 0) {
                free(pty->master.name);
                pty->master.name = NULL;
                close(pty->master.fd);
                free(pty->slave.name);
                pty->slave.name = NULL;
            }
            else break;
        }
    }

    return pty;
}

/*
 *----------------------------------------------------------------------
 * ttygetattr -
 *      return the attributes of a tty
 *----------------------------------------------------------------------
 */
TTYATTR *
```



```
ttygetattr(fd)
int fd;
{
    TTYATTR *ttyattrp;

    ttyattrp = (TTYATTR *) malloc(sizeof(TTYATTR));
    if (ttyattrp == 0) error_pep("ttygetattr: ioctl");

    if (ioctl(fd, TIOCGETP, (char *) &(ttyattrp->sgttyb)) < 0)
        error_pep("ttygetattr: ioctl");
    if (ioctl(fd, TIOCGETC, (char *) &(ttyattrp->tchars)) < 0)
        error_pep("ttygetattr: ioctl");
    if (ioctl(fd, TIOCGETD, (char *) &(ttyattrp->ldisc)) < 0)
        error_pep("ttygetattr: ioctl");
    if (ioctl(fd, TIOCGLTC, (char *) &(ttyattrp->ltchars)) < 0)
        error_pep("ttygetattr: ioctl");
    if (ioctl(fd, TIOCLGET, (char *) &(ttyattrp->lmask)) < 0)
        error_pep("ttygetattr: ioctl");

    return ttyattrp;
}

/*
 *----------------------------------------------------------------------
 * ttysetattr -
 *      set the attributes of a tty
 *----------------------------------------------------------------------
 */
int
ttysetattr(fd, ttyattrp)
int fd;
TTYATTR *ttyattrp;
{
    int rc;

    if (ioctl(fd, TIOCSETP, (char *) &(ttyattrp->sgttyb)) < 0)
        error_pen("ttygetattr: ioctl");
    if (ioctl(fd, TIOCSETC, (char *) &(ttyattrp->tchars)) < 0)
        error_pen("ttygetattr: ioctl");
    if (ioctl(fd, TIOCSETD, (char *) &(ttyattrp->ldisc)) < 0)
        error_pen("ttygetattr: ioctl");
    if (ioctl(fd, TIOCSLTC, (char *) &(ttyattrp->ltchars)) < 0)
        error_pen("ttygetattr: ioctl");
    if (ioctl(fd, TIOCLSET, (char *) &(ttyattrp->lmask)) < 0)
        error_pen("ttygetattr: ioctl");

    return 0;
}

/*
 *----------------------------------------------------------------------
```

```
 * ttycopyattr -
 *      copy the attributes of a tty
 *-----------------------------------------------------------------------
 */
int
ttycopyattr(fdfrom, fdto)
int fdfrom, fdto;
{
    TTYATTR *ttyattrp;
    int rc;

    ttyattrp = ttygetattr(fdfrom);
    if (ttyattrp == NULL) return NULL;
    rc = ttysetattr(fdto, ttyattrp);
    free(ttyattrp);
    return (rc < 0) ? -1 : 0;
}

/*
 *-----------------------------------------------------------------------
 * ttysetecho -
 *      set the echo on a tty
 *-----------------------------------------------------------------------
 */
int
ttysetecho(fd)
int fd;
{
    struct sgttyb sgttyb;

    if (ioctl(fd, TIOCGETP, (char *) &sgttyb) < 0)
        error_pen("ttysetecho: ioctl");
    sgttyb.sg_flags &= ~ECHO;
    if (ioctl(fd, TIOCSETP, (char *) &sgttyb) < 0)
        error_pen("ttysetecho: ioctl");
}

/*
 *-----------------------------------------------------------------------
 * ttysetcbreak -
 *      set CBREAK mode on a tty
 *-----------------------------------------------------------------------
 */
int
ttysetcbreak(fd)
int fd;
{
    struct sgttyb sgttyb;

    if (ioctl(fd, TIOCGETP, (char *) &sgttyb) < 0)
        error_pen("ttysetcbreak: ioctl");
    sgttyb.sg_flags |= CBREAK;
    if (ioctl(fd, TIOCSETP, (char *) &sgttyb) < 0)
        error_pen("ttysetcbreak: ioctl");
}

/*
 *-----------------------------------------------------------------------
 * ptyexecv -
 *      execute a program on a PTY
 *-----------------------------------------------------------------------
 */
int
ptyexecv(name, argv, pty, fds)
char *name;
char *argv[];
PTY *pty;
int fds;
{
    int child;
    int fd;

    if ((child = vfork()) < 0) error_pen("ptyexec: vfork");

    if (child == 0) {
        close(pty->master.fd);
        for (fd = 0; fds; fd++, fds >>= 1)
            if (fds & 01)
                if (dup2(pty->slave.fd, fd) < 0) error_pen("ptyexec: dup2");
        close(pty->slave.fd);
        execv(name, argv);
        perror("ptyexecv: execv");
        _exit(1);
    }
    else {
        close(pty->slave.fd);
        return(child);
    }
}
```

BLINDPRINT

makefile

```
MONDEF = MONITOR
MONINCLUDE = monitor
DEBUG = -DMONDEBUG -DDEBUG
CFLAGS = -D$(MONDEF) -Usun

blindprint:     blindprint.o monlib
        cc  -o blindprint blindprint.o monlib.ar

blindprint.o:   blindprint.c ../$(MONINCLUDE)/mondefs.h
        cc  -c blindprint.c

monlib: readrecord.o dumprecord.o printevents.o
        ar ru monlib.ar readrecord.o dumprecord.o printevents.o
        ranlib monlib.ar

readrecord.o:   readrecord.c ../$(MONINCLUDE)/montypes.h
        cc  -c readrecord.c

printevents.o:  printevents.c  ../$(MONINCLUDE)/mondefs.h \
                ../$(MONINCLUDE)/montypes.h
        cc  -c printevents.c

dumprecord.o:   dumprecord.c ../$(MONINCLUDE)/montypes.h
        cc  -c dumprecord.c
```

blindprint.c

```
/* blindprint.c - main */
/* monitor project */
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
 |                                                                              |
 |      project:        Monitor project under Richard Snodgras at UNC           |
 |                                                                              |
 |      programmer:     Stephen Duncan, 3/6/84                                  |
 |                                                                              |
 |      Blind print of sensor records.  Format of each record taken             |
 |      from the sensor descripter file with each int field in the              |
 |      descripter being a short in C, and each double int in the               |
 |      descripter file being an int in C.  The length field present            |
 |      in the second byte of each record is used to determine how              |
 |      much to read from the file.                                             |
 |                                                                              |
 |      Input:          stdin - sensor records from the accountant              |
 |                      or files named on command line                          |
 |                                                                              |
 |      Output:         stdout - ascii of fields in sensor record,              |
 |                          without filtering                                   |
 |                      stderr -                                                |
 |                          truncated sensor message                            |
 |                          invalid sensor id                                   |
 |                                                                              |
 |      Modules: main                                                           |
 |      External modules: <sys/montypes.h>,                                     |
 |                        readrecord.c                                          |
 |                        printevents.c                                         |
 |                        dumpevents.c                                          |
 |                                                                              |
 |      Maintenance:                                                            |
 |                                                                              |
 *!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/

#include        <stdio.h>
#include        <monitor/monops.h>
#include        <monitor/montypes.h>

/*
 *------------------------------------------------------------------------
 *   reads sensor record and
 *      prints ascii of each record, one per line.
 *------------------------------------------------------------------------
 */
main (argc, argv)
int     argc;
char  * argv[];
{
    mon_putevent record;                        /* event record */
    int     rstatus,                    /* status of calls */
            i;                          /* index for argv */
    FILE  * fp,                         /* for input */
          * fopen(),
          * fclose();

    if (argc == 1)
    {
```

```
    while ((rstatus = readrecord (stdin, &record)) != 0)
    /*
     * Note that the else condition (bad read)
     * just reads the next record
     */
        if (rstatus > 0)
            switch (record.cmd.type)
            {
                case MONOP_PUTEVENT_INT:
                    printint (stdout,&record);
                    break;

                case MONOP_PUTEVENT_EXT:
                    printext (stdout,&record);
                    break;

                default:
                /*
                 * This is probably just an error record
                 */
                    fprintf (stderr,
                            "%s: Bad command: %d at disp %d in stdin\n",
                            argv[0], record.cmd.type, ftell(stdin));
                    dumprecord (stderr, &record);
            }
            else                        /* an error */
                fprintf (stderr, "%s: Read error for stdin at %d\n",
                    argv[0], argv[i], ftell(stdin));
}
else
    for (i = 1; i < argc; i++)
    {
        if ((fp = fopen (argv[i], "r")) == NULL)
        {
            fprintf (stderr, "%s: Can't open %s\n", argv[0], argv[i]);
            exit (1);
        }
        else
            while ((rstatus = readrecord (fp, &record)) != 0)
            {
            /*
             * Note that the else condition (bad read)
             * just reads the next record
             */
                if (rstatus > 0)
                    switch (record.cmd.type)
                    {
                        case MONOP_PUTEVENT_INT:
                            printint (stdout,&record);
                            break;
                        case MONOP_PUTEVENT_EXT:
                            printext (stdout,&record);
                            break;

                        default:
                            fprintf (stderr,
                                "%s: Bad command: %d at disp %d in %s\n",
                                argv[0], record.cmd.type,
                                ftell(fp), argv[i]);
                            dumprecord (stderr, &record);
                    }
                else                        /* an error */
                    fprintf (stderr, "%s: Read error for file %s at %d\n",
                        argv[0], argv[i], ftell(fp));
            }

        fclose (fp);
    }
}
```

---

readrecord.c

```
/* readrecord.c - readrecord */

#include <stdio.h>
#include <monitor/montypes.h>
#define FIXEDLEN sizeof(struct mon_cmd)*2

/*------------------------------------------------------------------------
 * readrecord
 *      - reads an event record from a file pointer into a putevent struct
 *      - puts event record into structure pointed to by recd
 *      - returns the length of the record or -1 on an io error.
 *------------------------------------------------------------------------
 */
readrecord(fp,recd)
FILE    *fp;
mon_putevent *recd;
{
        int     stat;                   /* for io calls */
        int     varlen;                 /* length of record past cmd */
```

```
        short   *record=(short *)recd;  /* a little short hand */

        /*
         * Find length of record
         */

        if ( (stat = fread (record, sizeof(short), FIXEDLEN, fp)
             ) !=  FIXEDLEN
           )
                if (feof(fp))  return (0);      /* all done */
                else  return(-1);               /* some error */

        /*
         * Determine length in shorts remaining, (note recd==record)
         */

        varlen = (int)recd->cmd.length - FIXEDLEN;

        /*
         * Get remainder of record
         */

        if ( (stat = fread ((record + FIXEDLEN),
                            sizeof(short), varlen, fp)
             ) !=  varlen )
                return (-1);            /* didn't read whole record */

        return (recd->cmd.length);      /* return length in short ints */
}
```

---

printevents.c

```
/* printevents.c - printint printext */

#include <stdio.h>
#include <monitor/montypes.h>

/*
 *------------------------------------------------------------------
 * printint
 *      print internal event recds
 *------------------------------------------------------------------
 */

printint(fp,recd)
FILE                    * fp;
register mon_putevent * recd;
{
        int     index=0;

        switch(recd->eventnumber)
                {
                case 1:
                        fprintf (fp, "command = kernel\tlength = %6d\t",
                                recd->cmd.length);
                        fprintf (fp, "eventname = NameStart\t");
                        fprintf (fp, "performer = %6d\tobject = %d,%d\t",
                                recd->performer,
                                (recd->object)>>16,
                                (recd->object)&0xffff);
                        fprintf (fp, "initiator = %6d\ttimestamp = %10u\n",
                                recd->initiator,
                                recd->timestamp);
                        break;

                case 2:
                        fprintf (fp, "command = kernel\tlength = %6d\t",
                                recd->cmd.length);
                        fprintf (fp, "eventname = NextComponent\t");
                        fprintf (fp, "performer = %6d\tobject = %d,%d\t",
                                recd->performer,
                                (recd->object)>>16,
                                (recd->object)&0xffff);
                        fprintf (fp, "initiator = %6d\ttimestamp = %10u\t",
                                recd->initiator,
                                recd->timestamp);
                        fprintf (fp, "filename = %s\n",
                                recd->fields);
                        break;

                case 3:
                        fprintf (fp, "command = kernel\tlength = %6d\t",
                                recd->cmd.length);
                        fprintf (fp, "eventname = INodeCreate\t");
                        fprintf (fp, "performer = %6d\tobject = %d,%d\t",
                                recd->performer,
                                (recd->object)>>16,
                                (recd->object)&0xffff);
                        fprintf (fp, "initiator = %6d\ttimestamp = %10u\n",
                                recd->initiator,
                                recd->timestamp);
                        break;

                case 5:
                        fprintf (fp, "command = kernel\tlength = %6d\t",
```

```
                recd->cmd.length);
        fprintf (fp, "eventname = OpenSucessful\t");
        fprintf (fp, "performer = %6d\tobject = %d,%d\t",
                recd->performer,
                (recd->object)>>16,
                (recd->object)&0xffff);
        fprintf (fp, "initiator = %6d\ttimestamp = %10u\t",
                recd->initiator,
                recd->timestamp);
        fprintf (fp, "mode = %6d\tinitsize = %10d\n",
                recd->fields[0],
                *(int *)(recd->fields +1));
        break;

case 6:
        fprintf (fp, "command = kernel\tlength = %6d\t",
                recd->cmd.length);
        fprintf (fp, "eventname = FileClose\t");
        fprintf (fp, "performer = %6d\tobject = %d,%d\t",
                recd->performer,
                (recd->object)>>16,
                (recd->object)&0xffff);
        fprintf (fp, "initiator = %6d\ttimestamp = %10u\t",
                recd->initiator,
                recd->timestamp);
        fprintf (fp, "finalsize = %10d\n",
                *(int *)(recd->fields +0));
        break;

case 7:
        fprintf (fp, "command = kernel\tlength = %6d\t",
                recd->cmd.length);
        fprintf (fp, "eventname = INodeDelete\t");
        fprintf (fp, "performer = %6d\tobject = %d,%d\t",
                recd->performer,
                (recd->object)>>16,
                (recd->object)&0xffff);
        fprintf (fp, "initiator = %6d\ttimestamp = %10u\n",
                recd->initiator,
                recd->timestamp);
        break;

case 8:
        fprintf (fp, "command = kernel\tlength = %6d\t",
                recd->cmd.length);
        fprintf (fp, "eventname = ReadSensor\t");
        fprintf (fp, "performer = %6d\tobject = %d,%d\t",
                recd->performer,
                (recd->object)>>16,
                (recd->object)&0xffff);
        fprintf (fp, "initiator = %6d\ttimestamp = %10u\t",
                recd->initiator,
                recd->timestamp);
        fprintf (fp, "filepos = %10d\tactualcount = %6d\n",
                *(int *)(recd->fields +0),
                recd->fields[2]);
        break;

case 9:
        fprintf (fp, "command = kernel\tlength = %6d\t",
                recd->cmd.length);
        fprintf (fp, "eventname = WriteSensor\t");
        fprintf (fp, "performer = %6d\tobject = %d,%d\t",
                recd->performer,
                (recd->object)>>16,
                (recd->object)&0xffff);
        fprintf (fp, "initiator = %6d\ttimestamp = %10u\t",
                recd->initiator,
                recd->timestamp);
        fprintf (fp, "filepos = %10d\tactualcount = %6d\n",
                *(int *)(recd->fields +0),
                recd->fields[2]);
        break;

default:
        fprintf (stderr,
                "printevent: No internal eventnumber %5d\n",
                recd->eventnumber);
        dumprecord(stderr, recd);
        }
}

/*
 *----------------------------------------------------------------------
 * printext
 *      print external event recds
 *----------------------------------------------------------------------
 */

printext(fp,recd)
FILE            * fp;
register mon_putevent * recd;
{
        int     index=0;

        switch(recd->eventnumber)
                {
                case 0:
                        fprintf (fp, "command = external\tlength = %6d\t",
                                recd->cmd.length);
                        fprintf (fp, "eventname = AcctHeader\t");
                        fprintf (fp, "performer = %6d\tobject = 10%d\t",
```

```
                recd->performer,
                recd->object);
        fprintf (fp, "initiator = %6d\ttimestamp = %10u\t",
                recd->initiator,
                recd->timestamp);
        fprintf (fp, "acct date = 10%d\t",
                *(long *)(recd->fields + index));
                index += 2;
        fprintf (fp, "kernel date = 10%d\t",
                *(long *)(recd->fields + index));
                index += 2;
        fprintf (fp, "hostname = %s\t",
                (char *)(recd->fields + index));
        index += (strlen((char *)(recd->fields+index))+2)>>1;
        fprintf (fp, "init text = %s\n",
                (char *)(recd->fields + index));
        break;

case 1:
        fprintf (fp, "command = external\tlength = %6d\t",
                recd->cmd.length);
        fprintf (fp, "eventname = UserXamplSensor\t");
        fprintf (fp, "performer = %6d\tobject = 10%d\t",
                recd->performer,
                recd->object);
        fprintf (fp, "initiator = %6d\ttimestamp = %10u\t",
                recd->initiator,
                recd->timestamp);
        fprintf (fp, "str_parm_1 = %s\t",
                (char *)(recd->fields + index));
        index += (strlen((char *)(recd->fields+index))+2)>>1;
        fprintf (fp, "sh_parm_2 = 6%d\t",
                recd->fields[index++]);
        fprintf (fp, "str_parm_3 = %s\t",
                (char *)(recd->fields + index));
        index += (strlen((char *)(recd->fields+index))+2)>>1;
        fprintf (fp, "lg_parm_4 = 10%d\t",
                *(long *)(recd->fields + index));
        break;

case 2:
        fprintf (fp, "command = external\tlength = %6d\t",
                recd->cmd.length);
        fprintf (fp, "eventname = Header\t");
        fprintf (fp, "performer = %6d\tobject = %d,%d\t",
                recd->performer,
                (recd->object)>>16,
                (recd->object)&0xffff);
        fprintf (fp, "initiator = %6d\ttimestamp = %10u\t",
                recd->initiator,
                recd->timestamp);
                fprintf (fp, "header = %s\n",
                        (char *)(recd->fields));
                break;

        default:
                fprintf (stderr,
                        "printevent: No external eventnumber %5d\n",
                        recd->eventnumber);
                dumprecord(stderr, recd);
        }
}
```


---

dumprecord.c

```
/* dumprecord.c dumprecord */

#include <monitor/montypes.h>
#include <stdio.h>

/*
 *-----------------------------------------------------------------
 * dumprecord
 *      - dumps an event record in hex to a file descripter
 *-----------------------------------------------------------------
 */

dumprecord(file,record)
FILE     *file;
mon_putevent *record;
{
        unsigned short *ptr = (unsigned short *)&record;
        unsigned short *end = ptr + record->cmd.length;

        for ( ; ptr < end ; ptr++ )
                fprintf(file, "%04x ", *ptr);
        fprintf(file, "\n");
}
```

DESCHEMA

makefile

```
CFILES = deschema.c
N = /dev/null
OFILES = deschema.o
LIB = ../lib /usr/include/monitor
LIBFILES = ../lib/streamio.o ../lib/tuple.o ../lib/schema_idl.o \
        ../lib/readrecord.o ../lib/writerecord.o
CFLAGS = -g -I../lib -I/usr/include/monitor

deschema: $(OFILES) $(LIBFILES)
        cc -o deschema $(CFLAGS) $(OFILES) $(LIBFILES) \
                /usr/softlab/lib/libidl.a


depend:
        egrep "^#include" $(CFILES) $N | grep -v '<' | sed -e "/<.*>/d" \
        -e 's/:[        ]*\#include[    ]*\"\(.*\)\".*$$/:        \1/' \
        -e "s/\.c/.o/" > /tmp/dep
        -for i in `awk '{print $$2}' /tmp/dep | sort | uniq` ; \
        do for i in . $(LIB) ; \
                do if [ -f $$i/$$i ] ; \
                        then    echo  "s,$$i,$$i/$$i," ; \
                                break ; \
                        fi ; \
                done ; \
        done > /tmp/sedfile
        sed -f /tmp/sedfile /tmp/dep > /tmp/dep2
        sed -e '/^\# Dependencies/,$$ d' makefile > /tmp/makefile
        echo "# Dependencies DON'T REMOVE THIS LINE" \
                | cat - /tmp/dep2 >> /tmp/makefile
        mv makefile makefile.old
        cp /tmp/makefile makefile
        -rm -f /tmp/dep /tmp/dep2 /tmp/sedfile /tmp/makefile

# Dependencies DON'T REMOVE THIS LINE
deschema.o:     montypes.h
deschema.o:     ../lib/schema_idl.h
deschema.o:     ../lib/streamio.h
deschema.o:     ../lib/tuple.h
```

deschema.c

```
/* deschema.c - main */

#include <stdio.h>
#include "montypes.h"
#include "schema_idl.h"
#include "streamio.h"
#include "tuple.h"

/*
 *------------------------------------------------------------------------
 * main -
 *      splits off a schema from a stream and writes it to a file.
 *------------------------------------------------------------------------
 */

#define BUFSIZE 1024

char    *Toolname;                                      /* invoked name */

main(argc,argv)
int     argc;
char    **argv;
{
        FILE    *fp;                    /* ptr to file to receive schema */
        Mstream *sp;                    /* input stream                  */
        database schema;                /* schema from input             */
        int     io_count,               /* count read or written         */
                buffer[BUFSIZE];        /* io buffer                     */

        Toolname = *argv++;             /* save name for messages        */

        if ( argc != 2 )
        {
            fprintf(stderr, "Usage: %s filename\n", Toolname);
            exit(1);
        }
        if ( (fp = fopen(*argv,"w")) <= 0 )
        {
            perror("Opening schema file");
            exit(1);
        }

        if ( (sp = str_open(stdin)) == NULL )
        {
            fprintf(stderr, "%s: Can't open input stream\n", Toolname);
            exit(1);
```

```
    }
    if ( (schema = str_schemaread(sp)) == NULL )
    {
        fprintf(stderr, "%s: Can't read schema\n", Toolname);
        exit(1);
    }
    output(fp, schema);                    /* writes schem to file */
    fclose(fp);

    /*
     * Now we can just write as fast as we can.
     */

    while ( (io_count = fread(buffer, sizeof(int), BUFSIZE, stdin))
        > 0 )
    {
        if ( fwrite(buffer, sizeof(int), io_count, stdout) == 0  &&
                ferror(stdout) )
        {
            fprintf(stderr, "%s: Error writing event records\n",
                Toolname);
            exit(1);
        }
    }
    if ( ferror(stdin) )
    {
        fprintf(stderr, "%s: Error reading eventrecords\n", Toolname);
        exit(1);
    }
    exit(0);
}
```

ENSCHEMA

enschema.csh

```
#\!/bin/csh \-f
# Enschema - prepend a schema to a bunch of event records
# Usage:
#       enschema schemafile [-] [filename ...]
#
# Last Edit:    Mon Dec 2 16:41 1985
# Author:       Stephen E. Duncan <duncans@unc>
#               Department of Computer Science
#               University of North Carolina
#               Chapel Hill, NC  27514
#
# Copywrite (C) The University of North Carolina, 1985
#
#   All rights reserved. No part of this software may be sold or
#   distributed in any form or by any means without the prior written
#   permission of the SoftLab Software Distribution Coordinator.
#
#   Report problems to  softlab@unc (csnet) or
#                       softlab!unc@CSNET-RELAY (ARPAnet)
#   Direct all inquiries to the SoftLab Software Distribution
#       Coordinator, at the above addresses.
#

set rcsid='$Header:$'

#  Revision Log:
#       $Log:$
#
#  Edit Log:
#       Nov 20 1985 (duncans) Created.
#

if ( -e $1 ) then
        cat $*
        exit 0
else
        echo $0 : schema file $1 not found.
        exit 1
endif
```

FINITESTATE

makefile

```
FILES = main.c machineparse.y lex.l findpartition.c machine.c queue.c
OFILES = main.o machineparse.o findpartition.o machine.o queue.o lex.o
CFLAGS = -g -DYYDEBUG -I../lib -I/usr/include/monitor
LIBS = ../lib/libmontools.a -ll /usr/softlab/lib/libidl.a
YFLAGS = -d

finitestate:            $(OFILES) lex.l
        cc -o fs $(CFLAGS) $(OFILES) $(LIBS)

depend:
        egrep "^#include" $(FILES) | grep -v '<' | sed -e "/<.*>/d" \
        -e 's/:[        ]*\#include[     ]*\"\(.*\)\".*$$/:       \1/' \
        -e "s/\.c/.o/" > /tmp/dep
        -for i in `awk '{print $$2}' /tmp/dep | sort | uniq` ; \
        do for l in . $(LIB) ; \
                do if [ -f $$l/$$i ] ; \
                        then    echo  "s,$$i,$$l/$$i," ; \
                                break ; \
                        fi ; \
                done ; \
        done > /tmp/sedfile
        sed -f /tmp/sedfile /tmp/dep > /tmp/dep2
        sed -e '/^\# Dependencies/,$$ d' makefile > /tmp/makefile
        echo "# Dependencies DON'T REMOVE THIS LINE" \
                | cat - /tmp/dep2 >> /tmp/makefile
        mv makefile makefile.old
        cp /tmp/makefile makefile
        -rm -f /tmp/dep /tmp/dep2 /tmp/sedfile /tmp/makefile

# Dependencies DON'T REMOVE THIS LINE
main.o: schema_idl.h
main.o: tuple.h
main.o: streamio.h
main.o: ./finitestate.h
main.o: ./fstab.h
machineparse.y: schema_idl.h
machineparse.y: tuple.h
machineparse.y: ./finitestate.h
lex.l:  ./fs_machine.h
lex.l:  ./y.tab.h
findpartition.o:        schema_idl.h
findpartition.o:        tuple.h
findpartition.o:        ./finitestate.h
machine.o:      monops.h
machine.o:      montypes.h
machine.o:      schema_idl.h
machine.o:      streamio.h
machine.o:      tuple.h
machine.o:      ./finitestate.h
queue.o:        schema_idl.h
queue.o:        tuple.h
queue.o:        ./finitestate.h
queue.o:        montypes.h
```

finitestate.h

```
/* finitetypes.h - include file for finitemachine */

#include "tuple.h"
#include "fs_machine.h"                 /* for the fs_machine */

/*
 * Types for the partition
 */


typedef struct queue_node {             /* the tuples in sentence */
        tuple   *q_tuple;
        struct queue_node *q_next;
        } q_node;

typedef struct {                        /* queue of tuples in sentence */
        q_node  *q_head,
                *q_tail;
        int     q_count;
        } t_queue;

typedef struct {                        /* key to select partition struct */
        int     type;
        union   {
                char    *cval;
                long    nval;
                } u;
        } part_key;
```

```
typedef struct {                        /* status of partition's instance */
        part_key        p_key;
        t_queue         p_queue;
        int             p_state;
        } partition;

#define empty_q(queue_pointer) ((queue_pointer)->q_count == 0)
void insert_q();
tuple *delete_q();
```

fstab.h

```
/* fstab.h */

static char     rel_name[] = "FiniteState";          /* name of new rel */

/*
 * This is a table containing the attributes for the domains
 * of the FiniteState relation.
 */

struct fstab {
        char    name[20];       /* domain name                          */
        int     type,           /* integer, boolean, rational, string */
                length;         /* in bytes                             */
        } fstab[] = {
                { "cmdtype",        Ktype_integer, 1 },
                { "cmdlength",      Ktype_integer, 1 },
                { "eventnumber",    Ktype_integer, 2 },
                { "performer",      Ktype_integer, 2 },
                { "object",         Ktype_integer, 4 },
                { "initiator",      Ktype_integer, 2 },
                { "timestamp",      Ktype_integer, 4 },
                { "lasttimestamp",  Ktype_integer, 4 },
                { "partition",      Ktype_string, 24 },
                };
int fstabsize = sizeof(fstab) / sizeof(struct fstab);
```

machineparse.y

```
/* machineparse.y - yyparse, yyerror */


/*
 *----------------------------------------------------------------------
 * Parser for selection formula -
 *      This actually executes the formula for each record.
 *----------------------------------------------------------------------
 */

%{

#include <stdio.h>
#include "schema_idl.h"
#include "tuple.h"
#include "finitestate.h"

#ifdef TESTALONE


static char     Toolname[] = "test";          /* invoked name of program */

#else

extern char     *Toolname;                    /* invoked name of program */
extern machine  *Machine;

#endif TESTALONE

int             lineno = 0;                   /* line number of input    */


%}

%union  {
        char     *num;
        char     *name;
        testnode *testlist;
        linenode *line;
        };
%token  <name>  STRING
%token  <num>   NUM
%type   <name>          '-'
%type   <testlist>      test
```

```
%type   <testlist>      testlist
%type   <line>          line
%type   <line>          machine
%%
machine :       line                    { Machine = $1; $$ = Machine; }
        |       machine line            { if ( $1->l_state <= $2->l_state )
                                          {
                                                $1->l_next = $2;
                                                $$ = $2;
                                          }
                                          else
                                          {
                                                yyerror("States not in order")
;
                                                return(1);
                                          }
                                        }
        ;

line    :       NUM NUM testlist        { $$ = newline(atoi($1),atoi($2),$3);
}
        ;

testlist :      /* empty */             { $$ = NULL; }
        |       testlist test           { $$ = ($1)?
                                                ($1->t_next = $2, $$ = $1) :
                                                $2;
                                        }
        ;

test    :       STRING STRING NUM       { $$ = maketest($1, $2, $3); }
        |       STRING STRING STRING    { $$ = maketest($1, $2, $3); }
        |        '-'   STRING NUM       { $$ = maketest(NULL, $2, $3); }
        |        '-'   STRING STRING    { $$ = maketest(NULL, $2, $3); }
        |       STRING STRING  '-'      { $$ = maketest($1, $2, NULL); }
        |       STRING  '-'    '-'      { $$ = maketest($1, NULL, NULL); }
        |       error                   { return (1); }
        ;
%%

/*
 *----------------------------------------------------------------------
 * yyerror -
 *      routine invoked by parser when error is detected
 *----------------------------------------------------------------------
 */
void
yyerror(s)
char    *s;
{
        fprintf(stderr, "%s: yyparse: %s around line %d\n",
                Toolname, s, lineno);
}

/*
 *----------------------------------------------------------------------
 * newline -
 *      create and setup a new linenode in the machine
 *----------------------------------------------------------------------
 */
static linenode *
newline(state, nextstate, testlist)
int     state, nextstate;
testnode *testlist;
{
        linenode *lp = (linenode *)malloc(sizeof(linenode));

        lp->l_state = state;
        lp->l_nextstate = nextstate;
        lp->l_tlist = testlist;
        lp->l_next = NULL;
        return (lp);
}

/*
 *----------------------------------------------------------------------
 * maketest -
 *      create and setup new test condition
 *----------------------------------------------------------------------
 */
static testnode *
maketest(rel, dom, val)
char    *rel, *dom, *val;
{
        testnode *tp = (testnode *)malloc(sizeof(testnode));

        tp->t_next = NULL;
        if (rel)
        {
            tp->t_rel = (char *)malloc(strlen(rel)+1);
            strcpy(tp->t_rel,rel);
        }
        else tp->t_rel = NULL;
        if (dom)
        {
            tp->t_dom = (char *)malloc(strlen(dom)+1);
            strcpy(tp->t_dom,dom);
        }
        else tp->t_dom = NULL;
        tp->t_relop = '\0';
        if (val)
        {
```

```
        if ( *val == '\\' )
            switch(val[1])          /* check for relational operator */
            {
            case '!':               /* may be two position          */
                if (strlen(val)>3) break;
                switch(val[2])
                {
                case '=':
                case '>':
                case '<':
                case '\0':
                    tp->t_relop = val[1];
                    break;
                default:
                    ;
                }
                break;
            case '=':
            case '>':
            case '<':
                tp->t_relop = val[1];
                break;
            default:
                    ;               /* don't do anything            */
            }
        tp->t_val = (char *)malloc(strlen(val)+1);
        strcpy(tp->t_val,val);
    }
    else tp->t_val = NULL;
    return(tp);
} /* maketest */

#ifdef TESTALONE

/*
 * main -
 *      test routine
 */

main()
{
        int stat;
        stat = yyparse();
        exit(stat);
}

#endif TESTALONE
```


fs_machine.h

```
/* fs_machine.h - typedefs for the fs machine */

/*
 * The machine is built by the parser as linked lists.  To speed
 * execution, the linked lists are replaced by arrays terminated
 * by NULL pointers.
 */
#ifndef FS_MACHINE
#define FS_MACHINE

typedef struct tnode {                  /* condition in line */
        char            *t_rel,
                        *t_dom,
                        *t_val;
        char            t_relop;
        struct tnode    *t_next;
        } testnode;

typedef struct lnode {                  /* line (transition) in machine */
        int             l_state,
                        l_nextstate;
        testnode        *l_tlist;
        struct lnode    *l_next;
        } linenode, machine;

#endif
```

lex.l

```
/* lex.l - yylex */
%{
#include <stdio.h>
#include "fs_machine.h"
#include "y.tab.h"

/*
```

```
*-----------------------------------------------------------------------
* yylex -
*      lexical analyzer for finite state machine
*-----------------------------------------------------------------------
*/

extern char     *Toolname;      /* invoked name of program */
extern int      lineno;
char            nextchar;

%}

%%
[ \t]           { ; }   /* ignore white space */
\n              { lineno++; }
#.*$            { ; }   /* comment */
-?[0-9]+        {
                        yylval.num = (char *)malloc(yyleng+1);
                        strcpy(yylval.num, yytext);
                        return NUM;
                }
-               {
                        return yytext[0];
                }
\"[^"]+         {
                        if ( yytext[yyleng - 1] == '\\' )
                                yymore();       /* pick up the rest */
                        else if ( (nextchar = input()) )
                        {
                                yylval.name = (char *)malloc(yyleng);
                                strcpy(yylval.name, yytext+1);
                                yylval.name[strlen(yylval.name)] = '\0';
                                return STRING;
                        }
                        else return 0;
                }
[^- \t\n]+      {
                        yylval.name = (char *)malloc(yyleng+1);
                        strcpy(yylval.name, yytext);
                        return STRING;
                }
```


findpartition.c

```
/* findpartition.c - findpartition init_iterate iterate */
#include <stdio.h>
#include "schema_idl.h"
#include "tuple.h"
#include "finitestate.h"

/*
 *-----------------------------------------------------------------------
 *      This module implements an abstract structure containing a
 *      set of <key,value> pairs.
 *      The only operators are find and iterate.
 *      Find takes a key and searches the private structure for it,
 *      allocating a new one if not found.
 *-----------------------------------------------------------------------
 */
extern char     *Toolname;                      /* invoked name of program */
extern char     *part_name;                     /* name of partition       */

/*
 *-----------------------------------------------------------------------
 * partlist -
 *      private data structure containing a key and a value.
 *-----------------------------------------------------------------------
 */

partition *findpartition();
static partition *rfind();

static struct entry {
        part_key        key;                    /* key for this entry      */
        partition       part;                   /* values                  */
        struct entry    *next;                  /* linked list             */
        } *partlist = NULL;                     /* list of values          */

#define NUM 0                                   /* definitions for key.type */
#define CHAR 1

/*
 *-----------------------------------------------------------------------
 * find -
 *      search for the key in the data structure partlist,
 *      if one doesn't exist, add it.
 *      return a pointer to the partition in the entry
 *-----------------------------------------------------------------------
```

```
 */

partition *
findpartition(tp)
tuple   *tp;                                            /* input tuple */
{
        part_key        p_key;          /* key for partition into list */
        attribute       ap;             /* attributes of partition     */
        /*
         * find the partition in the tuple
         */

        if ( (ap = getdomainbyname(tp->relation->attributes,part_name))
            == NULL )
        {
            fprintf(stderr,"%s: record missing partition\n",Toolname);
            exit(1);
        }

        /*
         * Set up the key
         */

        switch (typeof(ap->attr_type))
        {
        case Ktype_string:
            p_key.type = CHAR;
            p_key.u.cval =
                (char *)malloc(strlen(&tp->record[ap->attr_pos]) +1 );
            strcpy(p_key.u.cval, &tp->record[ap->attr_pos]);
            break;
        case Ktype_boolean:
            p_key.type = NUM;
            p_key.u.nval = (long)(tp->record[ap->attr_pos] == 1);
            break;
        case Ktype_rational:        /* too tough to partition on a float */
            p_key.type = NUM;
            p_key.u.nval = (long)(*(float*)&tp->record[ap->attr_pos]);
            break;
        case Ktype_integer:
            p_key.type = NUM;
            switch(ap->attr_length)
            {
            case 1:
                p_key.u.nval = (int)(tp->record[ap->attr_pos]);
                break;
            case 2:
                p_key.u.nval = *(short*)&tp->record[ap->attr_pos];
                break;
            case 4:
                p_key.u.nval = *(long*)&tp->record[ap->attr_pos];
```



```
                break;
            default:
                fprintf(stderr,"%s: Invalid domain length for %s:%s\n",
                    Toolname, tp->relation->rel_name, ap->attr_name);
                exit(1);
            }
            break;
        default:
            fprintf(stderr,"%s: Invalid domain type for %s:%s\n",
                Toolname, tp->relation->rel_name, ap->attr_name);
            exit(1);
        }

        /*
         * Actually starts up recursive lookup with global partlist
         */

        return (rfind(&partlist,p_key));
} /* findpartition */
/*
 *----------------------------------------------------------------------
 * rfind -
 *      Does real work, traverses partlist looking for match on p_key
 *----------------------------------------------------------------------
 */
#define NEWENTRY (struct entry *)malloc(sizeof(struct entry))

static partition *
rfind(partlist,p_key)
struct entry    **partlist;             /* partition list, never NULL */
part_key        *p_key;                 /* lookup key                 */
{
        /*
         * This searches a sorted linked list for a key
         * and returns a pointer to the partition for that key.
         * The key may be numeric or character, and the entry
         * may have to be created.
         */

        int     sw;                         /* 3 way decision variable */
        register struct entry *ep;          /* pointer to list entry   */

        if ( *partlist != NULL)             /* Can still check         */
        {
            sw = (p_key->type == NUM) ?  /* put in form of strcmp      */
                p_key->u.nval - (*partlist)->key.u.nval
                : strcmp(p_key->u.cval,(*partlist)->key.u.cval);
            if ( sw == 0 )                  /* found it                */
                return(&((*partlist)->part));
            if ( sw > 0 )                   /* keep looking            */
                return(rfind((*partlist)->next,p_key));
```

```
        /*
         * If sw < 0, we should insert
         * the entry here, so just fall through.
         */
    }

    /*
     * Insert a new entry here
     * This was reached by either a null list
     * or the value is prior to the current list value.
     */
    if ( (ep = NEWENTRY) == NULL )          /* create a new entry  */
    {
        fprintf(stderr, "rfind: malloc failed\n");
        exit(1);
    }
    if (p_key->type == NUM)                 /* initialize p_key    */
    {
        ep->key.type = NUM;
        ep->key.u.nval = p_key->u.nval;
    }
    else                                    /* character key       */
    {
        ep->key.type = CHAR;
        ep->key.u.cval = (char *)malloc(strlen(p_key->u.cval) +1);
        strcpy(ep->key.u.cval, p_key->u.cval);
    }
    ep->part.p_state = 1;                   /* starting state      */
    ep->part.p_queue.q_head = NULL;         /* empty sentence      */
    ep->part.p_queue.q_tail = NULL;
    /*
     * Insert the new entry into the list
     */
    ep->next = *partlist;                   /* set up tail of list */
    *partlist  = ep;                        /* link into list      */
    return(&((*partlist)->part));
} /* rfind */
```

---

machine.c

```
/* machine.c - run accept reject */

#include <stdio.h>
#include "monops.h"
#include "montypes.h"
#include "schema_idl.h"
#include "streamio.h"
#include "tuple.h"
#include "finitestate.h"

machine         *Machine;                   /* the automata itself      */
extern char     *Toolname;                  /* invoked name of program  */
extern char     *part_name;                 /* name of partition domain */
extern relation fs_relation;                /* relation of partition    */
extern Mstream  *sp_out;                    /* output stream            */

/*
 *-------------------------------------------------------------------------
 * run -
 *      actually run the machine
 *-------------------------------------------------------------------------
 */

void
run(pp, tup_p)
partition *pp;                              /* struct for this partition */
tuple     *tup_p;                           /* input tuple               */
{
        int         state = pp->p_state,    /* current state             */
                    cmp;                    /* result of strncmp         */
        machine     *sp;                    /* pointer to state          */
        testnode    *tcp;                   /* test conditions           */
        SEQattribute seq_ap = tup_p->relation->attributes;
        attribute   ap,                     /* attributes of test domain */
                    o_ap;                   /* attributes of prev tuple  */
        char        *o_record;              /* record of prec tuple      */

        /*
         * Position at the right state
         */
        for ( sp = Machine; sp && sp->l_state != state; sp = sp->l_next )
                ;
        if ( sp == NULL || sp->l_state != state )
        {
            fprintf(stderr, "%s: panic in automata at %d.\n",
                Toolname, pp->p_state);
            exit(1);
        }
        /*
         * Check each line of the machine with the same state.
         * Take the newstate of the first line whose conditions are met.
         */
        for ( ; sp->l_state == state; sp = sp->l_next )
        {
```

```
/*
 * Go through the list of conditions, all of which must
 * me met to be accepted.
 */
for ( tcp = sp->l_tlist; tcp; tcp = tcp->t_next )
{
    if ( !(tcp->t_rel == NULL ||
        strcmp(tup_p->relation->rel_name,tcp->t_rel) == 0) )
        break;                          /* failed for this line */
    /*
     * We have matched the relation
     */
    if ( tcp->t_dom == NULL ) continue;
    else if ( (ap = getdomainbyname(seq_ap,tcp->t_dom)) == NULL )
        break;      /* failed the domain test */
    if ( tcp->t_val == NULL ) continue;
    if ( tcp->t_relop )             /* relational compare   */
    {
        /*
         * A relational operator compares to
         * the previous tuple in this partition.
         */
        if ( empty_q(&pp->p_queue) )
        {
            /*
             * Condition is always true the first time
             */
            insert_q(&pp->p_queue,tup_p);
            pp->p_state = sp->l_nextstate;
            return;
        }
        o_record = pp->p_queue.q_tail->q_tuple->record;
        o_ap = getdomainbyname(
            pp->p_queue.q_tail->q_tuple-> relation->attributes,
            tcp->t_dom);
        cmp = strncmp(tup_p->record[ap->attr_pos],
            o_record[o_ap->attr_pos],
            ap->attr_length);
        switch (tcp->t_relop)
        {
        case '!':               /* may be followed be relop */
            switch(tcp->t_val[1])
            {
            case '>':
                if ( !(cmp > 0) )   continue;
                else                break;
            case '<':
                if ( !(cmp < 0) )   continue;
                else                break;
            case '=':               /* These are synonymous */
            case '\0':
                if ( cmp != 0 )     continue;
                else                break;
            default:
                fprintf(stderr,"%s: corrupted test in state %d\n",
                    Toolname, pp->p_state);
                exit(1);
            }
            break;
        case '=':
            if ( cmp == 0 ) continue;
            else            break;
        case '>':
            if ( cmp > 0 )  continue;
            else            break;
        case '<':
            if ( cmp < 0 )  continue;
            else            break;
        }
    } /* if */
    else
        switch ( typeof(ap->attr_type) )
        {
        case Ktype_string:
            if ( strcmp(tup_p->record[ap->attr_pos],
                tcp->t_val) == 0 ) continue;
            else    break;
        case Ktype_boolean:
            if ( *tcp->t_val &&
                tup_p->record[ap->attr_pos] ) continue;
            else    break;
        default:
            if ( strncmp(tup_p->record[ap->attr_pos],
                o_record[o_ap->attr_pos],
                ap->attr_length) == 0 ) continue;
            else    break;
        }
} /* for */
if ( tcp == NULL )      /* unconditional, or conditions met */
{
    if ( sp->l_nextstate == 0 )                 /* accepted */
    {
            accept(pp,tup_p);
            return;
    }
    else if ( sp->l_nextstate < 0 )             /* rejected */
    {
            reject(pp);
            return;
    }
    else
    {
```

```
                /*
                 * To next state, so enqueue tuple, increment count,
                 * and set new state;
                 */

                insert_q(&pp->p_queue, tup_p);
                pp->p_state = sp->l_nextstate;          /* transition */
                return;
            }
        } /* if */
    } /* for */
    /*
     * No valid state was found, so reject the sentence
     */
    reject(pp);
    return;
} /* run */

/*
 *----------------------------------------------------------------------
 * accept -
 *      code for handling an accepted sentence
 *----------------------------------------------------------------------
 */

static
accept(pp, tup_p)
partition *pp;
tuple     *tup_p;
{
    tuple       *q_tup,   /* pointer to tuple in record queue      */
                fs_tup;   /* finite state tuple                    */
    mon_putevent fs_event; /* finite state event record            */
    SEQattribute tseq_ap; /* loop tmp                              */
    attribute    fs_ap,   /* attributes for domains in fs_tup      */
                 q_ap,    /* attributes for domains in tuple queue */
                 ap;      /* tmp                                   */

    /*
     * Create new tuple FINITESTATE and write it out
     */

    q_tup = pp->p_queue.q_head->q_tuple; /* needed to fill fs_event */
    fs_tup.relation = fs_relation;
    fs_tup.record = (char *)&fs_event;

    foreachinSEQattribute(fs_relation->attributes, tseq_ap, fs_ap)
    {
        if ( (q_ap = getdomainbyname(q_tup->relation, fs_ap->attr_name))
            == NULL)
            continue;
        else                                    /* we have a match */
            /*
             * This relies on all the fields in FiniteState being
             * at a fixed position and that equivalent fields
             * have the same length.  A straight struct copy
             * can't be used since the queue record may no
             * longer be in mon_putevent form.
             */
            strncpy((char *)&fs_tup.record[fs_ap->attr_pos],
                    &q_tup->record[ap->attr_pos],
                    ap->attr_length);
    }

    /*
     * Set up those domains unique to FiniteState and
     * that come from the last record.
     */

    fs_ap = getdomainbyname(q_tup->relation,"lasttimestamp");
    q_ap  = getdomainbyname(tup_p->relation,"timestamp");
    *(long *)&fs_tup.record[fs_ap->attr_pos] =
            *(long *)&tup_p->record[q_ap->attr_pos];

    fs_ap = getdomainbyname(fs_tup.relation,"partition");
    strcpy(&fs_tup.record[fs_ap->attr_pos], part_name);
    fs_ap->attr_length = strlen(part_name) + 1;
    fs_ap->attr_length += (fs_ap->attr_length % 2) ?
            0 : 1;

    fs_event.cmd.type = MONOP_PUTEVENT_EXT; /* not from kernel      */
    fs_event.cmd.length = (fs_ap->attr_pos + fs_ap->attr_length) >>1;
    fs_event.eventnumber = fs_relation->rel_sensor_id;

    str_write(sp_out, &fs_tup);

    while (! empty_q(&pp->p_queue) )        /* write out sentence   */
    {
            q_tup = delete_q(&pp->p_queue);
            str_write(sp_out, q_tup);
            free(q_tup);
    }
    str_write(sp_out, tup_p);               /* write out last tuple */
    free(tup_p);
    pp->p_state = 1;                        /* restart machine      */
}

/*
 *----------------------------------------------------------------------
 * reject -
 *      code for handling an rejected sentence
 *----------------------------------------------------------------------
```

```
 */
static
reject(pp)
partition       *pp;
{
        tuple   *q_tup;

        while (! empty_q(&pp->p_queue) )
        {
                q_tup = delete_q(&pp->p_queue);
                free(q_tup);
        }
        pp->p_state = 1;                                /* restart machine */
}
```

queue.c

```
/* queue.c - empty_q insert_q delete_q */

#include "schema_idl.h"
#include "tuple.h"
#include "finitestate.h"
#include "montypes.h"

/*
 *----------------------------------------------------------------------
 * insert_q -
 *      insert a new tuple into the queue
 *----------------------------------------------------------------------
 */

void
insert_q(qp, tup_p)
t_queue *qp;
tuple   *tup_p;
{
        q_node      *next = (q_node *)malloc(sizeof(q_node));

        /*
         * Set up q_node for current tuple
         */
        next->q_tuple = (tuple *)malloc(sizeof(tuple));
        next->q_tuple->relation = tup_p->relation;
        next->q_tuple->record = (char *)
            malloc(((struct mon_cmd *)tup_p->record)->length
            * 2 );                                      /* room for record */
        strncpy( next->q_tuple->record, tup_p->record,
            (((struct mon_cmd *)tup_p->record)->length * 2) );
        next->q_next = NULL;
        if ( empty_q(qp) )
        {
            qp->q_head = next;
            qp->q_tail = next;
        }
        else
            qp->q_tail->q_next = next;                  /* new tail      */
        qp->q_count++;                                  /* size of queue */
} /* insert_q */

/*
 *----------------------------------------------------------------------
 * delete_q -
 *      return the tuple at the tail of the queue
 *----------------------------------------------------------------------
 */
tuple *
delete_q(qp)
t_queue *qp;
{
        tuple   *tup_p;                     /* tuple to be returned    */
        q_node  *tmp;                       /* so we can free the node */

        if ( empty_q(qp) ) return(NULL);
        tup_p = qp->q_head->q_tuple;
        tmp = qp->q_head;
        qp->q_head = qp->q_head->q_next;
        free(tmp);
        qp->q_count--;
        return(tup_p);
} /* delete_q */
```

PROJECT

makefile

```
CFILES = main.c args.c modify_schema.c
OFILES = main.o args.o
LIB = ../lib /usr/include/monitor
LIBFILES = ../lib/streamio.o ../lib/tuple.o ../lib/schema_idl.o \
        ../lib/readrecord.o ../lib/writerecord.o
CFLAGS = -g  -I../lib -I/usr/include/monitor

project: $(OFILES) $(LIBFILES)
        cc -o project $(CFLAGS) $(OFILES) $(LIBFILES) \
                /usr/softlab/lib/libidl.a


depend:
        egrep "^#include" $(CFILES) | grep -v '<' | sed -e "/<.*>/d" \
                -e 's/:[        ]*\#include[    ]*\"\(.*\)\".*$$/:       \1/' \
                -e "s/\.c/.o/" > /tmp/dep
        -for i in `awk '{print $$2}' /tmp/dep | sort | uniq` ; \
        do for l in . $(LIB) ; \
                do if [ -f $$l/$$i ] ; \
                        then    echo  "s,$$i,$$l/$$i," ; \
                                break ; \
                        fi ; \
                done ; \
        done > /tmp/sedfile
        sed -f /tmp/sedfile /tmp/dep > /tmp/dep2
        sed -e '/^\# Dependencies/,$$ d' makefile > /tmp/makefile
        echo "# Dependencies DON'T REMOVE THIS LINE" \
                | cat - /tmp/dep2 >> /tmp/makefile
        mv makefile makefile.old
        cp /tmp/makefile makefile
#       -rm -f /tmp/dep /tmp/dep2 /tmp/sedfile /tmp/makefile

# Dependencies DON'T REMOVE THIS LINE
main.o: /usr/include/monitor/montypes.h
main.o: ../lib/schema_idl.h
main.o: ../lib/streamio.h
main.o: ../lib/tuple.h
main.o: ./project.h
args.o: ../lib/schema_idl.h
args.o: ../lib/tuple.h
args.o: ./project.h
modify_schema.o:        ../lib/schema_idl.h
modify_schema.o:        ../lib/tuple.h
modify_schema.o:        ./project.h
```

project.h

```
/* project.h */

/*
 * Options for project
 */

#define KEEP 1
```

main.c

```
/* main.c - main */

#include <stdio.h>
#include "montypes.h"
#include "schema_idl.h"
#include "streamio.h"
#include "tuple.h"
#include "project.h"

/*
 *------------------------------------------------------------------------
 * main -
 *      determine the relations and domains to project from
 *      the command line, read and revise the schema, read
 *      each record and reject or project as appropriate.
 *------------------------------------------------------------------------
 */

char    *Toolname;                              /* invoked name */
```

```
main(argc,argv)
int     argc;
char    **argv;
{
        int             options,                /* command line options   */
                        transfer_length,        /* how much to copy       */
                        stat;                   /* status of system calls */
        char            *inpos, *outpos;        /* positions in records   */
        Mstream         *sp_in, *sp_out;        /* i/o streams            */
        SEQrelation     proj_seq_rp;            /* relations to project   */
        tuple           tup_in, tup_out;        /* i/o tuples             */
        attribute       in_ap, out_ap;          /* specific domains       */
        SEQattribute    in_seq_ap, out_seq_ap,  /* domains for copy       */
                        t_seq_ap;               /* tmp for loop           */


        Toolname = *argv;                       /* save name for messages */

        if ( (sp_in = str_open(stdin)) == NULL )
        {
            fprintf(stderr, "%s: Can't open input stream\n", Toolname);
            exit(1);
        }

        if ( str_schemaread(sp_in) == NULL )    /* needed for args()      */
        {
            fprintf(stderr, "%s: Can't read input schema\n", Toolname);
            exit(1);

        }

        if ( (sp_out = str_open(stdout)) == NULL )
        {
            fprintf(stderr, "%s: Can't open output stream\n", Toolname);
            exit(1);
        }

        args(argc, argv, &options, &proj_seq_rp, sp_in->schema,
            &(sp_out->schema));                 /* handle command line  */
        tup_out.record = sp_out->record;

        while ( (stat = str_read(sp_in,&tup_in)) > 0 )
        {
            if ( ! inSEQrelation(proj_seq_rp,tup_in.relation) )
            {
                /*
                 * Code for relations not to be projected.
                 */

                if ( options&KEEP )             /* keep non-projected rels? */
```



```
                if ( (stat = str_write(sp_out,&tup_out)) < 0 )
                {
                    break;                      /* end on write error       */
                }
            else
                continue;                       /* next tuple               */
}

/*
 * Code for relations that are to be projected.
 *
 * Find the equivalent of the input relation in the output schema.
 * Transfer data between records in lumps of contiguous domains.
 * Write the new record.
 */

tup_out.relation = getrelationbysensorid(sp_out->schema,
    tup_in.relation->rel_sensor_id);    /* set output rel. */
outpos = tup_out.record; /* clear position in output record */
out_seq_ap = tup_out.relation->attributes;    /* init. list */

while ( ! emptySEQattribute(out_seq_ap) )
{
    /*
     * Transfer data between records.
     * Transfers equivalent contiguous domains as a lump.
     * Since order of domains may have changed, must start
     * looking at first input domain each time.
     * Must make sure that everything is properly aligned
     * on short boundaries.  Test for this before and after
     * finding lump.
     */

    retrievefirstSEQattribute(out_seq_ap, out_ap);
    in_seq_ap  =  tup_in.relation->attributes;
    foreachinSEQattribute(in_seq_ap, t_seq_ap, in_ap)
    {
        /*
         * Set in_seq_ap to first equiv dom. in out_seq_ap.
         * Note that it has to be there.
         */

        if ( strcmp(in_ap->attr_name, out_ap->attr_name) == 0 )
        {
            in_seq_ap = t_seq_ap;       /* 1st equiv. domain */
            inpos     = ( inpos < 0 ) ?    /* pos. in input */
                    tup_in.record - in_ap->attr_pos :
                    tup_in.record + in_ap->attr_pos;
            break;
        }
```

args.c

```
/* args.c - args */

#include <stdio.h>
#include "schema_idl.h"
#include "tuple.h"
#include "project.h"

/*
 *------------------------------------------------------------------------
 * args -
 *      Set flags and grab arguments from command line.
 *      Exits with 1 upon error.
 *------------------------------------------------------------------------
 */

extern char     *Toolname;      /* name under which program is invoked */

void
args(argc, argv, options, relations, schema, new_schema)
int         argc,               /* Number of arguments */
            *options;           /* command line options */
char        **argv;             /* Array of arguments */
SEQrelation *relations;         /* relations to be projected */
database    schema,             /* old schema */
            *new_schema;
{
        int         i,j,        /* Array subscripts */
                    offset = 0; /* position of current domain in relation */
        relation    new_rp = NULL,/* current relation in command line */
                    old_rp;     /* old version of relation */
        SEQattribute
                    tmp_seq_ap; /* tmp for foreach */
        attribute ap,           /* current attribute in command line */
                    old_ap,     /* old version of attribute */
                    tmp_ap;     /* temporary pointer */


        *options = 0;           /* Clear options */
        argv++;                 /* bypass program name */
        argc--;
        *new_schema = copyschema(schema);       /* set up new schema */
        initializeSEQrelation(*relations);      /* initialize sequence */
        for ( i = 0; i < argc; i++, argv++ )
        {
```



```
        if ( *(*argv+0) == '-' && *(*argv+1) )      /* test for option */
        {
            /*
             * Options can be individually specified or
             * all lumped together behind a single '-'
             */
            for ( j = 1; *(*argv + j) ; j++ )
                switch (*(*argv + j))
                {
                case 'k':
                    *options |= KEEP;
                    break;
                default:
                    fprintf(stderr, "Unknown option: %c\n",
                        *(*argv + j));
                    exit(1);
                } /* switch */
        } /* if */
        else                        /* select operand */
        {
            /*
             * This section sets the operands depending on
             * how many have been seen.  Note that storage
             * must be provided for each.
             */
            if ( new_rp == NULL || strcmp(*argv, "-") == 0 )
            {
                int d = 0;  /* count of domains */

                /*
                 * Set up new relation and append it
                 */

                if ( new_rp != NULL )       /* already seen one relation *

                {
                    new_rp->rel_vlensensor = ( offset < 0) ?
                        TRUE : FALSE;       /* handle last relation */
                    argv++;                 /* bypass "-" */
                }
                if ( (new_rp = getrelationbyname(*new_schema,*argv))
                    == NULL )
                {
                    fprintf(stderr, "%s: bad relation - %s\n",
                        Toolname, *argv);
                    exit(1);
                }
                new_rp->rel_name = (char *)GetHeap(strlen(*argv) + 1);
                strcpy(new_rp->rel_name, *argv);

                /*
```

```
}


/*
 * Note that an odd pos implies a 1 byte field
 * but an even pos can be any length.
 * Note that tup_in's attributes are all correct in pos,
 * but outpos has not been set yet.
 */
if (  (outpos - tup_out.record) % 2 == 1
   &&  (inpos - tup_in.record) % 2 == 0  )
{
    /*
     * Have to line things up. Other cases all work fine.
     * Worst case is a misaligned copy of lots of
     * 1 bytes fields.
     */

    if ( out_ap->attr_length != 1 )    /* next > 1 byte */
    {
        *outpos = *(outpos - 1);
                                /* align to even boundary */
        *(outpos++ - 1) = '\0';                 /* pad */
    }
    else                /* next domain is 1 byte long */
    {
        *outpos++ = *inpos;
        continue;                           /* next lump  */
    }
}

for (   transfer_length = 0;
        (! emptySEQattribute(out_seq_ap)  /* note break */
         && ! emptySEQattribute(in_seq_ap) );
        (out_seq_ap = tailSEQattribute(out_seq_ap),
         in_seq_ap  = tailSEQattribute(in_seq_ap) )
    )
{
    /*
     * Actually determine how much to copy at once.
     */

    retrievefirstSEQattribute(out_seq_ap, out_ap);
    retrievefirstSEQattribute(in_seq_ap, in_ap);
    if ( strcmp(out_ap->attr_name, in_ap->attr_name) == 0 )
        transfer_length += in_ap->attr_length;
    else
        break;                          /* end of series */
}
bcopy(inpos, outpos, transfer_length);
/*
```



```
         * Must make sure that everything is properly aligned
         * on short boundaries.
         */
        if ( ! transfer_length % 2 )          /* mis-aligned   */
        {
            /*
             * If the next domain doesn't take care of it,
             * a padding char must be added.
             */
            if ( (retrievefirstSEQattribute(
                tailSEQattribute(out_seq_ap), out_ap) ) == NULL
                ||  out_ap->attr_length != 1 )
            {
                /*
                 * This is the last domain.
                 */
                *(outpos + transfer_length - 1 ) =
                    *(outpos + transfer_length - 2);
                *(outpos++ + transfer_length - 1) = '\0';
            }
        }
        outpos += transfer_length;
        inpos  += transfer_length;
    } /* while */

    ((struct mon_cmd *)(tup_out.record))->length =
        (outpos - tup_out.record) / 2;   /* Reset record length */
    if ( (stat = str_write(sp_out,&tup_out)) < 0 )
        break;                          /* break loop on write err  */

} /* while */

if ( stat < 0 )                         /* see if here due to error */
{
    fprintf(stderr, "%s: I/O error in processing records\n",
        Toolname);
    exit(1);
}

exit(0);
} /* main */
```

modify_schema.c

```
/* modify_schema.c - modify_schema */

#include <stdio.h>
#include "schema_idl.h"
#include "tuple.h"
#include "project.h"

/*
 *----------------------------------------------------------------------
 * modify_schema -
 *      copy the schema and modify the copy to have
 *      the relations in seq_rp
 *----------------------------------------------------------------------
 */

extern  char    *Toolname;              /* invoked name from command line */

database
modify_schema(schema, seq_rp)
database        schema;         /* schema to be modified */
SEQrelation     seq_rp;         /* relations to be projected */
{
        SEQrelation     t_seq_rp;
        relation        old_rp,         /* tmp ptr to old entry */
                        new_rp,         /* new relation */
                        getrelationbyname();
        database        new_schema,     /* schema to be created */
                        copyschema();

        /*
         * Copy schema to new_schema and modify new_schema.
         */

        new_schema = copyschema(schema);

        /*
         * For each relation to be projected (in seq_rp), change its
         * entry in the schema.  Don't touch those not to be projected.
         * Must use name since copyschema avoids sharing.
         */

        foreachinSEQrelation(seq_rp, t_seq_rp, new_rp)
        {
                if ( (old_rp = getrelationbyname(new_schema, new_rp->rel_name)
                        != NULL )               /* see if it's there */
                        *old_rp = *new_rp;      /* replace it */

        } /* foreachinSEQ */
} /* modify_schema */
```

```
            * Add relation to list of relations to project,
            * must use schema instead of new_schema since
            * comparison is done on input.
            */

           old_rp = getrelationbyname(schema,*argv);
           appendrearSEQrelation(*relations,old_rp);

           /*
            * Set up fields in relation, don't touch
            * cmd.type, cmd.length, and eventnumber of
            * attributes. These are the first NUMFIXEDDOMS
            * domains.  Truncate rest of attributes.
            */
           new_rp->rel_sensor_id = old_rp->rel_sensor_id;
           new_rp->rel_vlensensor = FALSE;     /* initially */
           foreachinSEQattribute(new_rp->attributes, tmp_seq_ap, ap)
           {
               if ( ++d >= NUMFIXEDDOMS )
               {
                   tmp_seq_ap->next = NULL;    /* truncate list */
                   break;
               }
           }

           if ( (offset = ap->attr_pos) > 0 ) /* fixed position */
               offset += ap->attr_length;
       } /* if */
       else /* gather in domains */
       {
           /*
            * Note that this code allows for duplicate domains!
            * This is a feature, though maybe not that useful
            * a one.
            */
           if ( (tmp_ap = getdomainbyname(old_rp,*argv)) == NULL )
           {
                   fprintf(stderr, "%s: bad domain - %s\n",
                       Toolname, *argv);
                   exit(1);
           }
           if ( (ap = copyattribute(tmp_ap)) == NULL )
           {
                   fprintf(stderr, "%s: couldn\'t copy domain - %s\n"

                       Toolname, *argv);
                   exit(1);
           }
           appendrearSEQattribute(new_rp->attributes,ap);

           /*
            * Handle the new position
            */

           if ( offset < 0 )           /* variable length */
               ap->attr_pos = offset;  /* str_read() sets offset */
           else                        /* fixed length portion */
           {
               if ( typeof(ap->attr_type) == Ktype_string )
               {
                   /*
                    * Position becomes variable after this
                    */

                   ap->attr_pos = offset;
                   offset = -offset - 2;
                   new_rp->rel_vlensensor = TRUE;
               }
               else if ( ap->attr_length == 1 )
                   ap->attr_pos = offset++;
               else
               {
                   ap->attr_pos = (offset % 2 == 0) ?
                       offset : ++offset;      /* align */
                   offset += ap->attr_length;
               }
           } /* else  fixed length */
       } /* else  handle domains */
   } /* else   handle operand */
} /* for */

/*
 * Make sure that we have what is needed.
 * relation shouldn't be empty
 */
if ( new_rp != NULL )   /* we have something */
   new_rp->rel_vlensensor = ( offset < 0) ? TRUE : FALSE;
else
{
   fprintf(stderr,"%s: no relations to project\n",Toolname);
   exit(1);
}
} /* args */
```

```
};
#endif
```

---

main.c

```
/* main.c - main */

#include <stdio.h>
#include "schema_idl.h"
#include "streamio.h"
#include "tuple.h"
#include "y.tab.h"
#include "select.h"

/*
 *----------------------------------------------------------------------
 * main -
 *      process the args,
 *      read the schema
 *      check each record against formula for selection
 *----------------------------------------------------------------------
 */

char          *Toolname;            /* invoked name of program          */
tuple         tup;                  /* relation info and event recd     */
relation      rp;                   /* relation to select upon          */
bool          select;               /* boolean selection set by yyparse */
extern struct token_entry *formula_list;       /* lex. analyzed formula */

main(argc,argv)
int     argc;
char    **argv;
{
        int       stat,                     /* status of calls          */
                  parse_errs = 0;           /* count of bad parses      */
        char      *formula;                 /* selection formula        */
        Mstream   *sp_in, *sp_out;          /* input and output streams */
        struct token_entry *formula_head;   /* saved head of form. list */


        /*
         * Basic operation is to read the schema, determine the
         * relation to be selected upon, and run a parser on each
```



```
         * incoming record.  The parser sets the global "select"
         * to determine selection.
         */
        Toolname = *argv;                   /* save for error messages */
        if ( (sp_in = str_open(stdin)) == NULL )
        {
            fprintf(stderr, "%s: Can't open input stream \n", Toolname);
            exit(1);
        }
        if ( (sp_out = str_open(stdout)) == NULL )
        {
            fprintf(stderr, "%s: Can't open output stream \n", Toolname);
            exit(1);
        }
        if ( (sp_in->schema = str_schemaread(sp_in)) == NULL )
        {
            fprintf(stderr, "%s: Can't read input schema \n", Toolname);
            exit(1);
        }
        sp_out->schema = sp_in->schema;
        args(argc, argv, &rp, &formula, sp_in->schema);
        build_formula_list(formula);        /* lex. analysis of formula   */
        formula_head = formula_list;        /* save head of list          */
        while ( (stat=str_read(sp_in,&tup)) > 0 )
        {
            select  = FALSE;                /* initialize for this record */
            formula_list = formula_head;    /* reinitialize list          */
            if (rp)                         /* only on one relation       */
            {
                if ( rp == tup.relation )
                {
                    if ( yyparse() == 0 && select )
                        str_write(sp_out, &tup);
                    else
                        parse_errs++;
                }
            }
            else                            /* select on all relations */
            {
                if ( yyparse() == 0 && select )
                    str_write(sp_out, &tup);
                else
                    parse_errs++;
            }
        }
        if ( parse_errs )
        {
            fprintf(stderr, "%s: %d parsing errors encountered.\n",
                Toolname, parse_errs);
        }
        exit(0);
```

SELECT

makefile

```
CFILES = main.c args.c finddomain.c parse.y lex.c
OFILES = main.o args.o finddomain.o parse.o lex.o
LIB = ../lib /usr/include/monitor
LIBFILES = ../lib/streamio.o ../lib/tuple.o ../lib/schema_idl.o \
        ../lib/readrecord.o ../lib/writerecord.o
CFLAGS = -g -DYYDEBUG -I../lib -I/usr/include/monitor
YFLAGS = -d

select: $(OFILES) $(LIBFILES)
        cc -o select $(CFLAGS) $(OFILES) $(LIBFILES) \
                /usr/softlab/lib/libidl.a

depend:
        egrep "^#include" $(CFILES) | grep -v '<' | sed -e "/<.*>/d" \
                -e 's/:[        ]*\#include[    ]*\"\(.*\)\".*$$/:      \1/' \
                -e "s/\.c/.o/" > /tmp/dep
        -for i in `awk '{print $$2}' /tmp/dep | sort | uniq` ; \
        do for l in . $(LIB) ; \
                do if [ -f $$l/$$i ] ; \
                        then    echo  "s,$$i,$$l/$$i," ; \
                                break ; \
                        fi ; \
                done ; \
        done > /tmp/sedfile
        sed -f /tmp/sedfile /tmp/dep > /tmp/dep2
        sed -e '/^\# Dependencies/,$$ d' makefile > /tmp/makefile
        echo "# Dependencies DON'T REMOVE THIS LINE" \
                | cat - /tmp/dep2 >> /tmp/makefile
        mv makefile makefile.old
        cp /tmp/makefile makefile
#       -rm -f /tmp/dep /tmp/dep2 /tmp/sedfile /tmp/makefile

# Dependencies DON'T REMOVE THIS LINE
main.o: ../lib/schema_idl.h
main.o: ../lib/streamio.h
main.o: ../lib/tuple.h
main.o: ./y.tab.h
main.o: ./select.h
args.o: ../lib/schema_idl.h
args.o: ../lib/streamio.h
args.o: ../lib/tuple.h
finddomain.o:   ./select.h
finddomain.o:   ./y.tab.h
finddomain.o:   ../lib/schema_idl.h
finddomain.o:   ../lib/tuple.h
finddomain.o:   ../lib/streamio.h
parse.y:        ./select.h
lex.o:  ./y.tab.h
lex.o:  ./select.h
```

select.h

```
/* select.h - header file for select */

#define TRUE    1
#define FALSE   0

#define BOOLOPS "()|=><&|"                  /* operators in formula      */

#define LITERAL 1                           /* type of token_entry       */
#define LOOKUP  2
#define INLIST  3

typedef unsigned char bool;

struct token_entry {                        /* entry in process formula  */
        int     entry_type;                 /* LITERAL LOOKUP or INLIST  */
        int     token_type;                 /* not defined for NOTTOKEN  */
        union   {
                char    *lookup_name;       /* when lookup               */
                YYSTYPE inlist_lval;        /* when inlist               */
                int     literal_val;        /* when literal              */
                } v;
        struct token_entry *next;           /* next entry in table       */
        };


#ifdef OLDSTUFF
struct token_entry {                         /* entry in process formula */
        int     type;                        /* token type               */
        char    *domain_name;                /* name of domain if lookup */
        struct token_entry *next;            /* next entry in table      */
        YYSTYPE yylval;                      /* lval for parser          */
        bool    lookup;                      /* lookup or in yylval      */
```

```
                "%s: finddomain -- bad length for %s in %s: %d\n",
                Toolname, dom_name, tup.relation->rel_name,
                ap->attr_length);
            return(0);
        }
        return NUMBER;

    case(Ktype_rational):
        yylval.num = *(float *)(&tup.record[ap->attr_pos]);
        return NUMBER;

    case(Ktype_boolean):
        yylval.boolean = ( tup.record[ap->attr_pos] != 0 );
        return BOOLEAN;

    case(Ktype_string):
        yylval.str = tup.record + ap->attr_pos;
        return STRING;

    default:
        fprintf(stderr,
            "%s: finddomain -- bad type for %s in %s: %d\n",
            Toolname, dom_name, tup.relation->rel_name,
            typeof(ap->attr_type));
        return(0);
    } /* switch */
} /* finddomain */
```

---

parse.y

```
/* parse.y - yyparse, yyerror */

/*
 *----------------------------------------------------------------------
 * Parser for selection formula -
 *      This actually executes the formula for each record.
 *----------------------------------------------------------------------
 */

%union  {
        double          num;
        char            *str;
        char            *rexp;
        unsigned char   boolean;
        };

%{

#include "select.h"
#include <stdio.h>

extern char *Toolname;                  /* invoked name of program    */
extern bool select;                     /* set to determine selection */
char    *emsg,                          /* error message from re_comp */
        *re_comp();

%}
%token <num>    NUMBER          /* lexical analyzer interprets domains */
%token <str>    STRING
%token <rexp>   REGEXP
%token <boolean> BOOLEAN
%type   <boolean> bexpr expr numexpr strexpr
%left   '|'
%left   '&'
%left   '=' '<' '>'
%left   '!'
%%
/*
 * Boolean expression.
 *      handles parens, AND, OR, NOT, and errors
 */
bexpr   : expr                  { $$ = $1; select = $1; YYACCEPT;}
        | '!' expr              { if ( $2 == 0 ) $$ = 1; else $$ = 0; }
        | '!' bexpr             { $$ = ( $2 == 0 ); }
        | expr '|' expr         { if ( $1 || $3 ) $$ = 1; else $$ = 0; }
        | bexpr '|' bexpr       { $$ = ( $1 || $3 ); }
        | expr '&' expr         { if ( $1 && $3 ) $$ = 1; else $$ = 0; }
        | bexpr '&' bexpr       { $$ = ( $1 && $3 ); }
        | '(' expr ')'          { $$ = $2; }
        | '(' bexpr ')'         { $$ = $2; }
        | error                 { YYABORT; }
        ;

/*
 * Expression.
 *      union of numerical expression and string expression and
 *      use of BOOLEAN
 */
expr    : numexpr
        | strexpr
        | BOOLEAN               { $$ = $1; }
        | BOOLEAN '=' BOOLEAN   { if ( $1 == $3 ) $$ = 1; else $$ = 0; }
        | error         { YYABORT; }
        ;
```

```
} /* main */
```

args.c

```
/* args.c - args */

#include <stdio.h>
#include "schema_idl.h"
#include "streamio.h"
#include "tuple.h"

/*
 *------------------------------------------------------------------------
 * args -
 *      process arguments from command line
 *------------------------------------------------------------------------
 */

extern  char *Toolname;

args(argc, argv, rp, formula, schema)
int      argc;
char     **argv,
         **formula;
relation *rp;
{
        if ( argc > 3 )
        {
            fprintf(stderr, "Usage: %s relation formula\n", Toolname);
            exit(1);
        }
        if ( (*rp = getrelationbyname(schema, *++argv)) == NULL )
        {
            fprintf(stderr, "%s: relation not in schema\n", Toolname);
            exit(1);
        }
        *formula = *++argv;
} /* args */
```

finddomain.c

```
/* finddomain.c - finddomain */

#include <stdio.h>
#include "y.tab.h"
#include "select.h"
#include "schema_idl.h"
#include "tuple.h"
#include "streamio.h"

/*
 * finddomain -
 *      find the domain in the tuple and set yylval accordingly
 *      return a token based on the attr_type
 *
 */
extern tuple    tup;            /* input tuple                          */
extern char     *Toolname;      /* invoked name of program              */

finddomain(dom_name)
char    *dom_name;                              /* name of the domain   */
{
        attribute       ap;                     /* attributes of domain */

        if ( (ap = getdomainbyname(tup.relation, dom_name)) == NULL )
            return(0);
        switch ( typeof(ap->attr_type) )
        {
        case(Ktype_integer):
            switch(ap->attr_length)
            {
            case 1:
                yylval.num = (float)
                    ((unsigned char)tup.record[ap->attr_pos]);
                break;
            case 2:
                yylval.num = (float)
                    *(short *)(&tup.record[ap->attr_pos]);
                break;
            case 4:
                yylval.num = (float)
                    *(int *)(&tup.record[ap->attr_pos]);
                break;
            default:
                fprintf(stderr,
```

```
        entry = (struct token_entry *)malloc(sizeof(struct token_entry));
        if ( formula_list == NULL )      /* starting out            */
        {
            formula_list = entry;
            list_tail    = entry;
        }
        else
            list_tail->next = entry;     /* add at tail of list     */
        entry->next = NULL;

        /*
         * Determine entry characteristics.
         */
        if ( (index("-+",*for_p) && isdigit(*(for_p+1)))
            || isdigit(*for_p)
            || (*for_p == '.'  && isdigit(*(for_p+1)) )   )
        {
            /*
             * Numeric token
             */
            entry->entry_type = INLIST;
            if ( sscanf(for_p,"%f",(float *)&f_tmp) == 1 )
            {
#ifdef YYDEBUG
                fprintf(stderr,
                    "%s: build_formula_list -- num = %f\n",
                    Toolname, f_tmp);
#endif YYDEBUG
                /*
                 * Point for_p to after number.
                 */
                entry->v.inlist_lval.num = f_tmp;
                for_p++;                     /* bypass possible sign */
                while( isdigit(*for_p) || *for_p == '.' ) for_p++;
                if ( *for_p == '.' )
                    while ( isdigit(*++for_p) );
                if ( *for_p == 'e' || *for_p == 'E' )
                {
                    for_p++;                 /* bypass possible sign */
                    while ( isdigit(*++for_p) );
                }
                entry->token_type = NUMBER;
                list_tail = entry;
            }
            else
                return(1);                   /* couldn't read it     */
        }
        else if ( *for_p == '/' )
        {
            /*
             * Regular Expression
             */
            entry->entry_type = INLIST;
            str_p = ++for_p;                 /* hang on to start */
            for ( ; !(for_p >= for_end || *for_p == '/'); for_p++)
            {
                if ( *for_p == '\\' )        /* escape character */
                    for_p++;
            }
            entry->v.inlist_lval.rexp =
                (char *)malloc(for_p - str_p + 1);
            strncpy(entry->v.inlist_lval.rexp, str_p, for_p - str_p);
            entry->token_type = REGEXP;
            list_tail = entry;
        }
        else if ( *for_p == '"' )
        {
            /*
             * String
             */
            entry->entry_type = INLIST;
            str_p = ++for_p;                 /* hang on to start */
            for ( ; !(for_p >= for_end || *for_p == '\"'); for_p++)
            {
                if ( *for_p == '\\' )        /* escape character */
                    for_p++;
            }
            entry->v.inlist_lval.str =
                (char *)malloc(for_p - str_p + 1);
            entry->token_type = STRING;
            strncpy(entry->v.inlist_lval.str, str_p, for_p - str_p);
            list_tail = entry;
        }
        else if ( index(BOOLOPS,*for_p) )        /* operator    */
        {
            entry->entry_type = LITERAL;
            entry->v.literal_val = (int)*for_p++;
            list_tail = entry;               /* point to tail of list */
        }
        else if ( isalpha(*for_p) || *for_p == '_' ) /* domain name */
        {
            if ( (*for_p == 'T' || *for_p == 'F')    /* boolean?    */
                && (index(BOOLOPS,*(for_p+1)) || *(for_p+1) == '\0') )
            {
                entry->entry_type = INLIST;
                entry->token_type = BOOLEAN;
                entry->v.inlist_lval.boolean = (*for_p == 'T');
                list_tail = entry;     /* point to tail of list     */
            }
            else
            {
                for ( str_p = for_p;
```

```
/*
 * Numerical expression
 *      any expression with NUMBERs
 */
numexpr : NUMBER '=' NUMBER     { if ( $1 == $3 ) $$ = 1; else $$ = 0; }
        | NUMBER '<' NUMBER     { if ( $1 < $3 ) $$ = 1; else $$ = 0; }
        | NUMBER '>' NUMBER     { if ( $1 > $3 ) $$ = 1; else $$ = 0; }
        | '(' numexpr ')'       { $$ = $2; }
        | error         { YYABORT; }
        ;

/*
 * String expression
 *      logical expressions with strings and regular expressions
 */
strexpr : STRING '=' STRING     {if (strcmp($1,$3) == 0) $$ = 1; else $$ = 0;
}
        | STRING '=' REGEXP     {if ( (emsg=re_comp($3)) == NULL ) $$ = re_exe
c($1,$3); else $$ = 0; }
        | STRING '<' STRING     {if (strcmp($1,$3) < 0) $$ = 1; else $$ = 0; }
        | STRING '<' REGEXP     {if (strcmp($1,$3) < 0) $$ = 1; else $$ = 0; }
        | STRING '>' STRING     {if (strcmp($1,$3) > 0) $$ = 1; else $$ = 0; }
        | STRING '>' REGEXP     {if (strcmp($1,$3) > 0) $$ = 1; else $$ = 0; }
        | '( ' strexpr ')'      { $$ = $2; }
        | error         { YYABORT; }
        ;
%%

/*
 *----------------------------------------------------------------------
 * yyerror -
 *      routine invoked by parser when error is detected
 *----------------------------------------------------------------------
 */
void
yyerror(s)
char    *s;
{
/* #include <signal.h> */

        fprintf(stderr, "%s: yyparse: %s\n", Toolname, s);
        /* kill(0,SIGQUIT); */
}
```


lex.c

```
/* lex.c - build_formula_list yylex */

#include <stdio.h>
#include <ctype.h>
#include "y.tab.h"
#include "select.h"

extern char *Toolname;              /* invoked name of program          */
struct token_entry *formula_list;   /* list of tokens in formula        */

/*
 *----------------------------------------------------------------------
 * build_yylex_list -
 *      create a list of the tokens in the formula, uses global
 *      formula_list, formula, and
 *----------------------------------------------------------------------
 */

build_formula_list(for_p)
char    *for_p;                             /* formula from command line */
{
        int     i;                          /* index into dom_name       */
        float   f_tmp;                      /* hold numeric from formula */
        char    *str_p,                     /* start of string           */
                *for_end = for_p + strlen(for_p);     /* end of formula */
        struct token_entry *entry,          /* list entry                */
                           *list_tail;      /* head of list              */

        /*
         * Process til end of formula.
         * Add entry to front of list.
         */

        formula_list = NULL;                /* initialize list           */
        while ( for_p < for_end )
        {
            while ( isspace(*for_p) && for_p < for_end )
                    for_p++;                /* ignore white space        */
            if ( for_p >= for_end || *for_p == '\0' )
                    return(0);              /* all done                  */
            /*
             * Allocate and initialize a new entry.
             */
```

makefile

```
CFILES = main.c
OFILES = main.o
CFLAGS = -g $(LIBS)
LIB = ../lib ../include /usr/include/monitor
LIBS = -I../lib -I../include -I/usr/include/monitor
LIBES = ../lib/libmontools.a /usr/softlab/lib/libidl.a

streamprint:    $(OFILES)
        cc $(CFLAGS) $(OFILES) -o streamprint $(LIBES)


depend:
        -rm -f /tmp/dep
        egrep "^#include" $(CFILES) /dev/null | grep -v '<' | sed -e "/<.*>/d"
\
                -e 's/:[        ]*\#include[     ]*\"\(.*\)\".*$$/:       \1/' \
                -e "s/\.c/.o/" > /tmp/dep
        -rm -f /tmp/sedfile
        touch /tmp/sedfile
        -for i in `awk "{print $$2}" /tmp/dep` ; \
        do for l in $(LIB) ; \
                do if [ -f $$l/$$i ] ; \
                        then    echo  >> /tmp/sedfile "s,$$i,$$l/$$i," ; \
                                break ; \
                        fi ; \
                done ; \
        done
        sed -f /tmp/sedfile /tmp/dep > /tmp/dep2
        sed -e '/^\# Dependencies/,$$ d' makefile > /tmp/makefile
        echo "# Dependencies DON'T REMOVE THIS LINE" \
                | cat - /tmp/dep2 >> /tmp/makefile
        mv makefile makefile.old
        cp /tmp/makefile makefile
        -rm -f /tmp/dep /tmp/dep2 /tmp/sedfile /tmp/makefile

# Dependencies DON'T REMOVE THIS LINE
main.o: ../lib/schema_idl.h
main.o: ../lib/tuple.h
main.o: ../lib/streamio.h
```

main.c

```
/* main.c - main */

#include <stdio.h>
#include "schema_idl.h"
#include "tuple.h"
#include "streamio.h"


/*
 *------------------------------------------------------------------------
 * main -
 *      print out in ascii the values for each record, one per line.
 *      Label the values if invoked with -l, toggled with -u.
 *------------------------------------------------------------------------
 */

char *Toolname;                         /* invoked name of program  */

main(argc, argv)
int  argc;
char *argv[];
{
        int         i,                  /* loop index               */
                    options = PRINTLABELS, /* command line options  */
                    stat;               /* status of system calls   */
        Mstream     *sp;                /* input stream             */
        tuple       tup;                /* input tuple              */

        /*
         * Process coommand line.
         */
        Toolname = *argv;               /* save for error messages  */
        while (++argv,--argc)
        {
            if ((*argv)[0] == '-')
            {
                for (i=1; (*argv)[i]; i++)
                {
                    switch((*argv)[i])
                    {
                    case 'l':
                        options = PRINTLABELS;   /* NOTE : this toggles */
                        break;
                    case 'u':
                        options = DONTPRINTLABELS;
                        break;
```

```
                    (!isspace(*for_p))
                        && (index(BOOLOPS,*for_p) == NULL);
                    for_p++)
                    ;
                entry->v.lookup_name =
                    (char *)malloc(for_p - str_p + 1);
                strncpy(entry->v.lookup_name, str_p, for_p - str_p);
                entry->entry_type = LOOKUP;       /* run time eval. */
                list_tail = entry;          /* point to tail of list */
            }
        }
        else                                /* literal               */
        {
            entry->entry_type = LITERAL;
            entry->v.literal_val = (int)*for_p++;
            list_tail = entry;              /* point to tail of list */
        }
    } /* while */
} /* build_formula_list */

/*
 *---------------------------------------------------------------------
 * Lexical analyzer for select formula -
 *      also does lookups into current tuple
 *---------------------------------------------------------------------
 */

yylex()
{
    int     type;                           /* token type          */

    if ( formula_list == NULL )             /* no more tokens      */
        return(0);
    else if ( formula_list->entry_type == LOOKUP )
        type = finddomain(formula_list->v.lookup_name);
    else if ( formula_list->entry_type == INLIST )
    {
        type   = formula_list->token_type;
        yylval = formula_list->v.inlist_lval;
    }
    else                                    /* must be LITERAL     */
        type = formula_list->v.literal_val;
    formula_list = formula_list->next;      /* point to next token */
    return(type);
} /* yylex */
```

SHUTDOWNACCT

makefile

```
MONDEF = MONITOR
MONINCLUDE = monitor
DEBUG = -DMONDEBUG -DDEBUG
CFLAGS = -D$(MONDEF) -Usun

shutdownacct: shutdownacct.c ../$(MONINCLUDE)/montypes.h \
        ../$(MONINCLUDE)/mondefs.h
        cc -g -D$(MONDEF) -o shutdownacct shutdownacct.c
```

shutdownacct.c

```
/* shutdownacct.c - main shutdown*/
/*
 *----------------------------------------------------------------------
 * shutdownacct
 *      - allows superuser to close down monitoring in case
 *      - of emergency.
 *      - Note: SYSL_MONITOR actually checks for superuser
 *----------------------------------------------------------------------
 */
#define MONITOR
#include <sys/syslocal.h>
#include <monitor/mondefs.h>
#include <monitor/montypes.h>
#include <stdio.h>


short           buffer[4096];
main ()
{
        shutdown();                     /* all done; clean up */
}

/*
 *----------------------------------------------------------------------
 * shutdown
 *      - closes down monitor
 *      - caller must be root
 *----------------------------------------------------------------------
 */
shutdown()
{
        struct mon_cmd command;
        int     i;

        command.type = MONOP_SHUTDOWN;
        command.length  = sizeof(struct mon_cmd);
        i = syscall(SYSLOCAL,SYSL_MONITOR, (unsigned char *)&command);
        fprintf(stderr,"shutdownacct: shutdown -- %d\n",i);
}
```

```
                default:
                    fprintf(stderr, "Usage: %s [-lu]\n", Toolname);
                    exit(1);
                }
            } /* for */
        } /* if */
        else
        {
            /*
             * try to open *argv as a stream (str_fopen) and
             * then print it.  treat '-' as stdin.
             * allow changing to unlabelled "in medias res"
             */
#ifdef MULTIFILE
            if ( strcmp(*argv,"-") == 0 )
            {
                sp = str_open(stdin);
                *argv = "standard input";      /* for error messages */
            }
            else if ( (sp = str_fopen(*argv,"r")) == NULL )
            {
                fprintf(stderr, "%s: can't open %s\n",
                    Toolname, *argv);
                continue;                      /* next argument      */
            }
            while ( (stat = str_read(sp,&tup)) > 0 )
                if ( tupleprint(stdout, &tup, options) != 0 )
                    fprintf(stderr,
                        "%s: can't print tuple in %s.\n",
                        Toolname, *argv);
            switch(stat)
            {
            case STRIO_ESCHEMA:
                fprintf(stderr, "%s: no schema for input.\n",
                    Toolname);
                exit(1);
                break;
            case STRIO_EREAD:
                fprintf(stderr, "%s: input read error.\n",
                    Toolname);
                exit(1);
                break;
            default:
                if ( stat < 0 )
                {
                    perror("streamprint: Reading");
                    exit(1);
                }
            }
#else
            fprintf(stderr, "Usage: %s [-lu]\n", Toolname);
                exit(1);
#endif MULTIFILE
            } /* else */
        } /* while */

#ifndef MULTIFILE
        sp = str_open(stdin);

        while ( (stat = str_read(sp,&tup)) > 0 )
            if ( tupleprint(stdout, &tup, options) != 0 )
            {
                fprintf(stderr, "%s: can't print tuple.\n", Toolname);
                exit(1);
            }
        switch(stat)
        {
        case STRIO_ESCHEMA:
            fprintf(stderr, "%s: no schema for input.\n",
                Toolname);
            exit(1);
            break;
        case STRIO_EREAD:
            fprintf(stderr, "%s: input read error.\n",
                Toolname);
            exit(1);
            break;
        default:
            if ( stat < 0 )
            {
                perror("streamprint: Reading");
                exit(1);
            }
        }
#endif MULTIFILE
        exit(0);
} /* main */
```

schema_idl.h

```
/* schema_idl.h - IDL declarations generated by idlc, version 2.0
    on Tue Apr  1 19:29:02 1986
*/
# include "/usr/softlab/include/C/global.h"

/* Private Types */

/* Class Headers */

/* Nodes */
typedef struct Rattribute * attribute;
# define Kattribute 2
# define Nattribute (_attribute((attribute)N_INIT( \
        GetNode(sizeof(struct Rattribute),Kattribute), \
        Kattribute,sizeof(struct Rattribute))))
# define Fattribute(n) (Xattribute(n); FreeNode(n, Kattribute);)
# define _attribute(N) (N)
# define Xattribute(N)

typedef struct Rdatabase * database;
# define Kdatabase 4
# define Ndatabase (_database((database)N_INIT( \
        GetNode(sizeof(struct Rdatabase),Kdatabase), \
        Kdatabase,sizeof(struct Rdatabase))))
# define Fdatabase(n) (Xdatabase(n); FreeNode(n, Kdatabase);)
# define _database(N) (N)
# define Xdatabase(N)

typedef struct Rrelation * relation;
# define Krelation 6
# define Nrelation (_relation((relation)N_INIT( \
        GetNode(sizeof(struct Rrelation),Krelation), \
        Krelation,sizeof(struct Rrelation))))
# define Frelation(n) (Xrelation(n); FreeNode(n, Krelation);)
# define _relation(N) (N)
# define Xrelation(N)

typedef int type_boolean;
# define Ktype_boolean 1
# define Ntype_boolean Ktype_boolean
# define Ftype_boolean(n)

typedef int type_integer;
# define Ktype_integer 3
# define Ntype_integer Ktype_integer
# define Ftype_integer(n)

typedef int type_rational;
# define Ktype_rational 5
# define Ntype_rational Ktype_rational
# define Ftype_rational(n)

typedef int type_string;
# define Ktype_string 7
# define Ntype_string Ktype_string
# define Ftype_string(n)


/* Classes */
typedef union {
    int IDLinternal;
    HgenericHeader IDLclassCommon;
    type_integer Vtype_integer;
    type_rational Vtype_rational;
    type_string Vtype_string;
    type_boolean Vtype_boolean;
} type;


/* Sets and Sequences */
typedef struct IDLtag1{
        struct IDLtag1 *next;
        relation value;
} Crelation, *Lrelation;

# define SEQrelation Lrelation
# define inSEQrelation(relationseq,relationvalue) IDLInList((pGenList)relation
seq,relationvalue)
# define initializeSEQrelation(relationseq) relationseq = NULL
# define appendfrontSEQrelation(relationseq,relationvalue) relationseq=\
                (SEQrelation)IDLListAddFront((pGenList)relationseq,relationval
ue)
# define appendrearSEQrelation(relationseq,relationvalue) relationseq=\
                (SEQrelation)IDLListAddRear((pGenList)relationseq,relationvalu
e)
# define orderedinsertSEQrelation(relationseq,relationvalue,relationcompfn) re
lationseq=\
                (SEQrelation)IDLListOrderedInsert((pGenList)relationseq,relati
onvalue,relationcompfn)
# define retrievefirstSEQrelation(relationseq, relationvalue)\
                relationvalue = (relation)IDLListRetrieveFirst((pGenList)relat
ionseq)
# define retrievelastSEQrelation(relationseq, relationvalue)\
                relationvalue = (relation)IDLListRetrieveLast((pGenList)relati
onseq)
```

LIBMONTOOLS

makefile

```
CFILES = streamio.c tuple.c readrecord.c writerecord.c
OFILES = streamio.o schema_idl.o tuple.o readrecord.o writerecord.o
INCLUDE = -I/usr/softlab/include
LIB =
LIBES = /usr/softlab/lib/libidl.a
IDLC = /usr/softlab/bin/idlc
IDLCFLAGS = -g
CFLAGS = -g $(INCLUDE)

libmontools.a: $(OFILES)
        ar ruc libmontools.a $(OFILES)
        ranlib libmontools.a

        $(IDLC) $(IDLCFLAGS) $<

schema_idl.h: schema_idl.o

depend:
        -rm -f /tmp/dep
        egrep "^#include" $(CFILES) | grep -v '<' | sed -e "/<.*>/d" \
                -e 's/:[        ]*\#include[    ]*\"\(.*\)\".*$$/:       \1/' \
                -e "s/\.c/.o/" > /tmp/dep
        -rm -f /tmp/sedfile
        touch /tmp/sedfile
        -for i in `awk "{print $$2}" /tmp/dep` ; \
        do for l in $(LIB) ; \
                do if [ -f $$l/$$i ] ; \
                        then    echo  >> /tmp/sedfile "s,$$i,$$l/$$i," ; \
                                break ; \
                        fi ; \
                done ; \
        done
        sed -f /tmp/sedfile /tmp/dep > /tmp/dep2
        sed -e '/^\# Dependencies/,$$ d' makefile > /tmp/makefile
        echo "# Dependencies DON'T REMOVE THIS LINE" \
                | cat - /tmp/dep2 >> /tmp/makefile
        mv makefile makefile.old
        cp /tmp/makefile makefile
        -rm -f /tmp/dep /tmp/dep2 /tmp/sedfile /tmp/makefile

# Dependencies DON'T REMOVE THIS LINE
streamio.o:     schema_idl.h
streamio.o:     streamio.h
streamio.o:     tuple.h
tuple.o:        schema_idl.h
tuple.o:        tuple.h
tuple.o:        tuple_key.h
tuple.o:        printlabels.h
```

schema_idl.idl

```
Structure schema Root database Is

        database        =>      database_name   :       String,
                                relations       :       Seq Of relation;

        relation        =>      rel_name        :       String,
                                rel_sensor_id   :       Integer,
                                rel_viensensor  :       Boolean,
                                        -- true if variable length
                                attributes      :       Seq Of attribute;

        attribute       =>      attr_name       :       String,
                                attr_length     :       Integer, -- in bytes
                                attr_pos        :       Integer,
                                        -- from beginning, < 0 if notfixed
                                attr_type       :       type;

        type            ::=     type_integer | type_rational |
                                type_string  | type_boolean;

        For type Use Enumerated;
--      type_integer=>; type_rational=>; type_string=>; type_boolean=>;


End

Process schema_idl Inv schema Is

Pre   input   :    schema;
Post  output  :    schema;

End
```

```
#define SCHEMAREAD      1
#define SCHEMAWRITTEN   2

#define STRIO_ESCHEMA   -1
#define STRIO_EREAD     -2

#ifndef MAXRECSIZE              /* to allow an override */
#define MAXRECSIZE      512
#endif

typedef struct S_Mstream {
        FILE    *fp;
        char    record[MAXRECSIZE];
        database schema;
        short   flag;
        } Mstream;

/*
 * Function declarations
 */

Mstream         *str_open(),
                *str_fopen();

database        str_schemaread();

int             str_schemawrite(),
                str_read(),
                str_write();

#endif STREAMIOINCLUDE
```

---

streamio.c

```
/* streamio.c - library routines for streamio in Monitor system
 *      str_open, str_fopen, str_schemaread, str_schemawrite,
 *      str_read, str_write
 */

#include <stdio.h>
#include <monitor/monops.h>
#include <monitor/montypes.h>
#include "schema_idl.h"
#include "streamio.h"
#include "tuple.h"

static Mstream stream[_NFILE];        /* has same index as fp into _iob */


/*
 *----------------------------------------------------------------------
 * str_open -
 *      associate a fp, gotten from fopen or somewhere,  with a Stream
 *      There is a one to one relationship between streams and FILEs
 *----------------------------------------------------------------------
 */
Mstream *
str_open(fp)
FILE    *fp;
{
        register Mstream *sp = NULL;          /* pointer to stream       */

        if ( fp == NULL )
        {
                fprintf(stderr, "str_open: null FILE pointer\n");
                return(NULL);
        }
        else
        {
                int             i;            /* index into record       */

                sp      = &stream[fp-stdin]; /* same stream for each fp */
                sp->fp  = fp;
                for ( i = 0; i < MAXRECSIZE; i++)       /* clear record */
                        sp->record[i] = NULL;
                sp->schema = NULL;
                sp->flag   = 0;               /* records schema status   */
                return(sp);
        }
} /* str_open */

/*
 *----------------------------------------------------------------------
 * str_fopen -
 *      open the named file and associate a Stream with it.
 *----------------------------------------------------------------------
 */
Mstream *
str_fopen(filename,mode)
char    *filename, *mode;
{
        register Mstream *sp = NULL;
        FILE             *fp;
```

```
# define ithinSEQrelation(relationseq, index, relationvalue)\
                relationvalue = (relation)IDLListRetrieveIth((pGenList)relatio
nseq, index)
# define tailSEQrelation(relationseq)\
                ((relationseq) ? relationseq->next : NULL)
# define removefirstSEQrelation(relationseq) relationseq=\
                (SEQrelation)IDLListRemoveFirstCell((pGenList)relationseq)
# define removeSEQrelation(relationseq,relationvalue) relationseq=\
                (SEQrelation)IDLListRemoveCell((pGenList)relationseq,relationv
alue)
# define removelastSEQrelation(relationseq) relationseq=\
                (SEQrelation)IDLListRemoveLastCell((pGenList)relationseq)
# define foreachinSEQrelation(relationseq,relationptr,relationvalue) for\
(relationptr = relationseq; \
                relationptr!=NULL&&((relationvalue=relationptr->value)||1); \
relationptr=relationptr->next)
# define emptySEQrelation(relationseq) ((relationseq)==NULL)
# define lengthSEQrelation(relationseq) IDLListLength(relationseq)

typedef struct IDLtag2{
        struct IDLtag2 *next;
        attribute value;
} Cattribute, *Lattribute;

# define SEQattribute Lattribute
# define inSEQattribute(attributeseq,attributevalue) IDLInList((pGenList)attri
buteseq,attributevalue)
# define initializeSEQattribute(attributeseq) attributeseq = NULL
# define appendfrontSEQattribute(attributeseq,attributevalue) attributeseq=\
                (SEQattribute)IDLListAddFront((pGenList)attributeseq,attribute
value)
# define appendrearSEQattribute(attributeseq,attributevalue) attributeseq=\
                (SEQattribute)IDLListAddRear((pGenList)attributeseq,attributev
alue)
# define orderedinsertSEQattribute(attributeseq,attributevalue,attributecompfn
) attributeseq=\
                (SEQattribute)IDLListOrderedInsert((pGenList)attributeseq,attr
ibutevalue,attributecompfn)
# define retrievefirstSEQattribute(attributeseq, attributevalue)\
                attributevalue = (attribute)IDLListRetrieveFirst((pGenList)att
ributeseq)
# define retrievelastSEQattribute(attributeseq, attributevalue)\
                attributevalue = (attribute)IDLListRetrieveLast((pGenList)attr
ibuteseq)
# define ithinSEQattribute(attributeseq, index, attributevalue)\
                attributevalue = (attribute)IDLListRetrieveIth((pGenList)attri
buteseq, index)
# define tailSEQattribute(attributeseq)\
                ((attributeseq) ? attributeseq->next : NULL)
# define removefirstSEQattribute(attributeseq) attributeseq=\
                (SEQattribute)IDLListRemoveFirstCell((pGenList)attributeseq)
# define removeSEQattribute(attributeseq,attributevalue) attributeseq=\
                (SEQattribute)IDLListRemoveCell((pGenList)attributeseq,attribu
tevalue)
# define removelastSEQattribute(attributeseq) attributeseq=\
                (SEQattribute)IDLListRemoveLastCell((pGenList)attributeseq)
# define foreachinSEQattribute(attributeseq,attributeptr,attributevalue) for\
(attributeptr = attributeseq; \
                attributeptr!=NULL&&((attributevalue=attributeptr->value)||1);
 \
attributeptr=attributeptr->next)
# define emptySEQattribute(attributeseq) ((attributeseq)==NULL)
# define lengthSEQattribute(attributeseq) IDLListLength(attributeseq)


/* Class Attributes */

/* Node Structures*/
struct Rattribute { IDLnodeHeader IDLhidden;
    String attr_name;
    int attr_length;
    int attr_pos;
    type attr_type;
};
struct Rdatabase { IDLnodeHeader IDLhidden;
    String database_name;
    SEQrelation relations;
};
struct Rrelation { IDLnodeHeader IDLhidden;
    String rel_name;
    int rel_sensor_id;
    Boolean rel_vlensensor;
    SEQattribute attributes;
};

/* Port Declarations */
void output();
database input();
```

---

streamio.h

```
/* streamio.h - header for using streamio */
#ifndef STREAMIOINCLUDE
#define STREAMIOINCLUDE         /* to prevent re-including */
```

```
        }

        /*
         * Set up tuple
         */
        tp->record = sp->record;
        tp->relation = getrelation(tp->record, sp->schema);
        setposition(tp);    /* this updates phisical position of domain */

        return(stat);
} /* str_read */

/*
 *----------------------------------------------------------------------
 * str_write -
 *      write out a tuple for this stream
 *----------------------------------------------------------------------
 */
int
str_write(sp,tp)
Mstream *sp;                            /* pointer to stream        */
tuple   *tp;                            /* pointer to tuple         */
{
        if ( !(sp->flag & SCHEMAWRITTEN) )  /* ensure schema is written */
            if ( sp->schema )
                (void)str_schemawrite(sp);  /* always ok.           */
            else
                return(STRIO_ESCHEMA);      /* must have a schema   */
        return( writerecord(sp->fp, (mon_putevent *)(tp->record)) );
} /* str_write */
```

tuple.h

```
/* tuple.h - header for using tuples */

#ifndef TUPLEINCLUDE
#define TUPLEINCLUDE

#define NUMFIXEDDOMS    3               /* the number of fixed domains  */
#define connectSEQattribute(s1,s2) s1 = \
(SEQattribute)IDLListConnect((pGenList)s1,(pGenList)s2)

#define PRINTLABELS     1               /* options for tupleprint       */
#define DONTPRINTLABELS 0

pGenList IDLListConnect();

typedef struct s_tuple {
        char            *record;        /* usually stream's rec.        */
        relation        relation;
        /*SEQattribute  domains;*/      /* same as relation->attributes */
        } tuple;


/*
 * Function declarations
 */

relation        getrelation(),
                getrelationbysensorid(),
                getrelationbyname(),
                copyrelation();

void            setpostion(),
                rmrelationbyname(),
                rmdomainbyname();

attribute       getdomainbyname(),
                copyattribute();

database        copyschema();

#endif TUPLEINCLUDE
```

tuple key.h

```
/* tuple_key.h */

/*
 * These are the key domains which must be in every event record
 */

#define KEYNAMESIZE     25      /* longest key name allowed */
static char key_attr_tab[][KEYNAMESIZE] = {
                "cmdtype",
```

```
        if ( (fp = fopen(filename,mode)) == NULL )
                ;                               /* just leave errno set */
        else
                sp = str_open(fp);
        return(sp);
} /* str_fopen */

/*
 *----------------------------------------------------------------------
 * str_schemaread -
 *      read in a schema from a stream
 *----------------------------------------------------------------------
 */
database
str_schemaread(sp)
Mstream *sp;
{
        int     c;          /* actually a char, but fgetc needs an int */

        if ( sp->flag & SCHEMAREAD)
        {
                fprintf(stderr,
                        "str_schemaread: schema already read\n");
                return(NULL);
        }
        if ( (sp->schema = input(sp->fp)) != NULL)
                sp->flag |= SCHEMAREAD;
        /*
         * There may be a \n left after the schema, but we want
         * to point to the beginning of the event records.
         */
        if ( (c=fgetc(sp->fp)) != '\n' )
                ungetc((char)c, sp->fp);       /* wasn't there afterall */
        return(sp->schema);
} /* str_schemaread */


/*
 *----------------------------------------------------------------------
 * str_schemawrite -
 *      write out the schema for this stream
 *----------------------------------------------------------------------
 */
int
str_schemawrite(sp)
Mstream *sp;
{
        if ( sp->flag & SCHEMAWRITTEN)
        {
                fprintf(stderr,
                        "str_schemawrite: schema already written\n");
```



```
                return(-1);
        }
        output(sp->fp,sp->schema);
        sp->flag |= SCHEMAWRITTEN;
        return(0);
} /* str_schemawrite */

/*
 *----------------------------------------------------------------------
 * str_read -
 *      read a tuple from the stream
 *----------------------------------------------------------------------
 */
int
str_read(sp,tp)
Mstream *sp;                                    /* pointer to stream */
tuple   *tp;                                    /* pointer to tuple  */
{
        int             stat=0;                 /* status of calls   */

        /*
         * Read in a schema, if necessary.
         */
        if ( !(sp->flag & SCHEMAREAD) )
        {
                if ( str_schemaread(sp) == NULL )
                {
                        sp->flag |= STRIO_ESCHEMA;
                        return(STRIO_ESCHEMA);
                }
        }

        /*
         * Read in the event record.
         */
        switch (stat = readrecord(sp->fp, (mon_putevent *)sp->record))
        {
            case -1:                            /* readrecord error     */
                return(STRIO_EREAD);
                break;
            case 0:                             /* EOF                  */
                return(stat);
                break;
            default:
                if ( stat > MAXRECSIZE )        /* Probably already an error */
                {
                        fprintf(stderr, "str_read: buffer overflow\n");
                        exit(1);
                }
```

```
            fprintf(stderr,"str_read: invalid relation %d\n", sensorid);
            return(NULL);
        }
        return(cur_rp);

} /* getrelationbysensorid */

/*
 *----------------------------------------------------------------------
 * getrelation -
 *      return a pointer to the relation in the record
 *----------------------------------------------------------------------
 */

relation
getrelation(record,schema)
short   *record;
database schema;
{
        register int sensorid = ((mon_putevent *)record)->eventnumber;

        return( getrelationbysensorid(schema, sensorid) );

} /* getrelation */

/*
 *----------------------------------------------------------------------
 * getrelationbyname -
 *      return a pointer (which may be NULL) to the relation with name
 *      in the schema
 *----------------------------------------------------------------------
 */
relation
getrelationbyname(schema,name)
database        schema;
char            *name;
{
        register relation       rp = NULL;
        register SEQrelation    tmp_seq_rp;          /* tmp in foreach */

        foreachinSEQrelation(schema->relations, tmp_seq_rp, rp)
                if ( strcmp(rp->rel_name, name) == 0 ) break;
        if (tmp_seq_rp != NULL)                      /* we found it    */
                return(rp);
        else
                return(NULL);                        /* didn't find it */
} /* getrelationbyname */

/*
 *----------------------------------------------------------------------
 * rmrelationbyname -
 *      remove the named relation from the schema, if it's there
 *----------------------------------------------------------------------
 */
void
rmrelationbyname(schema, name)
database        schema;
char            *name;
{
        register relation rp;                        /* pointer to relation */

        if ( (rp = getrelationbyname(schema,name)) != NULL )
                removeSEQrelation(schema->relations, rp);

} /* rmrelationbyname */

/*
 *----------------------------------------------------------------------
 * setposition -
 *      Set the position of each domain in the relation.
 *      Fixed length relations don't change, nor do the
 *      fixed fields of variable length relations.  A variable
 *      position field is indicated with a negative position.
 *----------------------------------------------------------------------
 */
void
setposition(tp)
tuple   *tp;
{
        SEQattribute tmp_ap;          /* pointer for foreach            */
        relation     rp = tp->relation;
        attribute    cur_ap;          /* pointer for foreach            */
        char         *buffer = tp->record;
        int          offset = 0,      /* accumulated offset into record */
                     slen;            /* intermediate value             */

        /*
         * Set up the attributes so that the position is correct for
         * the current sensor.  Only the positions for the variable
         * part of variable length sensors need be calculated.
         */
        if ( rp->rel_vlensensor )
        {
            foreachinSEQattribute(tp->relation->attributes,tmp_ap,cur_ap)
            {
                if (offset < 0)
                {
                    /*
                     * Variable position
                     */
                    if ( cur_ap->attr_length != 1 )
                        offset -= (-offset)%2;       /* align the domain */
```

```
                "cmdlength",
                "eventnumber"
                };
int     key_attr_tabsize = sizeof(key_attr_tab)/KEYNAMESIZE;
```

printlabels.h

```
/* printlabels.h */

/*
 * The label is the attr_name of the domain, the value is the value
 * field, except for boolean fields, which are T or F.  The attr_type
 * and lengeth are used to select the right format.
 */

#define FMT_RELATION    0
#define FMT_STRING      1
#define FMT_BOOLEAN     2
#define FMT_INT_1       3
#define FMT_INT_2       4
#define FMT_INT_4       5
#define FMT_RATIONAL    6

static char *l_format[] = {                     /* labelled format    */
        "%s::\t",                               /* relation name      */
        "%s = %s\t",                            /* type_string        */
        "%s = %s\t",                            /* type_boolean       */
        "%s = %-3d\t",                          /* type_int, length 1 */
        "%s = %-5d\t",                          /* type_int, length 2 */
        "%s = %-10ld\t",                        /* type_int, length 4 */
        "%s = %-10.10f\t",                      /* type_rational      */
        };

/*
 * The '%.0s' is used to discard an argument, so that the print routine
 * doesn't care about formatting.
 */
static char *u_format[] = {                     /* unlabelled format  */
        "%.0s",                                 /* relation name      */
        "%.0s%s\t",                             /* type_string        */
        "%.0s%s\t",                             /* type_boolean       */
        "%.0s%-3d\t",                           /* type_int, length 1 */
        "%.0s%-5d\t",                           /* type_int, length 2 */
        "%.0s%-10ld\t",                         /* type_int, length 4 */
        "%.0s%-10.10f\t",                       /* type_rational      */
        };
```

tuple.c

```
/* tuple.c - getrelation, getrelationbyname, rmrelationbyname,
 *      getdomainbyname, rmdomainbyname, setposition,
 *      copyschema, copyrelation, copyattribute,
 *      tupleprint ;
 *      tuple manipulation routines
 */

#define LIBTUPLE                /* signifies what to ignore in tuple.h */
#include <stdio.h>
#include <monitor/montypes.h>
#include "schema_idl.h"
#include "tuple.h"
#include "tuple_key.h"
#include "printlabels.h"

/*
 *----------------------------------------------------------------------
 * getrelationbysensorid -
 *      return a pointer to a relation in a database from a sensorid
 *----------------------------------------------------------------------
 */
relation
getrelationbysensorid(schema, sensorid)
database        schema;
int             sensorid;
{
        SEQrelation rp = schema->relations,     /* allowable relations */
                    tmp_rp;                     /* ptr for foreach     */
        relation    cur_rp;                     /* ptr for foreach     */

        foreachinSEQrelation(rp, tmp_rp, cur_rp)
        {
            if ( cur_rp->rel_sensor_id == sensorid )
                break;
        }
        if (tmp_rp == NULL)                             /* didn't find it */
        {
```

```
{
        register attribute new_ap = Nattribute;         /* new attribute */

        /*
         * All that needs to be done is to copy in the fields.
         * Note that name shares same space as old name.  This allows
         * comparisons on name pointer for similar domains
         */

        new_ap->attr_name   = ap->attr_name;
        new_ap->attr_length = ap->attr_length;
        new_ap->attr_pos    = ap->attr_pos;
        new_ap->attr_type   = ap->attr_type;
        return(new_ap);
} /* copyattribute */

#ifdef OBSOLETE
/*
 *----------------------------------------------------------------------
 * copykey -
 *      make a new copy of the key and all its parts
 *----------------------------------------------------------------------
 */

key
copykey(kp)
key     kp;             /* the key to be copied */
{
        key             new_kp = Nkey;          /* copy of key */
        SEQattribute    tail_seq_ap;            /* used in appending */
        register SEQattribute tmp_seq_ap;       /* for use in foreach */
        attribute       ap,                     /* pos in sequence */
                        new_ap;                 /* copy of attribute */

    /*
     * Copying consists of copying the name into a new area,
     * and copying the sequence of attributes.
     */
        if ( kp == NULL )       /* don't have to do anything */
                return( NULL );

        if ( (new_kp->key_name = (char *)GetHeap(strlen(kp->key_name) +1))
                == NULL )
        {
                /*
                 * No room left, we're in trouble
                 */
                fprintf(stderr, "copykey: no room in heap\n");
                return(NULL);
        }
        strcpy(new_kp->key_name, kp->key_name);

        if ( emptySEQattribute(kp->attributes) )        /* don't have to copy
 */
                return(new_kp);

    /*
     * Copy the attributes (all are immutable domains)
     */


        retrievefirstSEQattribute(kp->attributes, ap);
        if ( (new_ap = copyattribute(ap)) == NULL )     /* didn't work */
        {
                fprintf(stderr,"copykey: couldn't copy attributes\n");
                return(NULL);
        }
        appendfrontSEQattribute(new_kp->attributes, new_ap);

        tail_seq_ap = tailSEQattribute(kp->attributes);
        foreachinSEQattribute(tail_seq_ap, tmp_seq_ap, ap)
        {
                if ( (new_ap = copyattribute(ap)) == NULL ) /* didn't work */
                {
                        fprintf(stderr,"copykey: couldn't copy attributes\n");
                        return(NULL);
                }
                appendrearSEQattribute(new_kp->attributes, new_ap);
        }
    /*
     * All done.
     */
        return(new_kp);
} /* copykey */
#endif OBSOLETE

/*
 *----------------------------------------------------------------------
 * copyrelation -
 *      create a new copy in memory of the relation
 *----------------------------------------------------------------------
 */
relation
copyrelation(rp)
relation        rp;
{
        int             l;                      /* line in table      */
        relation        new_rp = Nrelation;     /* copy of rp         */
        SEQattribute    tail_seq_ap;            /* used in appending  */
        register        SEQattribute tmp_seq_ap; /* for use in foreach */
        attribute       ap = NULL,              /* pos in sequence    */
                        new_ap;                 /* copy of attribute  */
```

```
            if ( typeof(cur_ap->attr_type) == Ktype_string )
            {
                /*
                 * Length must be multiple of 2, since
                 * sensors write data in shorts.
                 */
                slen = strlen(buffer+offset) +1;
                cur_ap->attr_length = (slen%2)?
                    slen + 1  :  slen;
            }
            cur_ap->attr_pos = offset;
            offset -= cur_ap->attr_length;
        } /* if */
        else
        {
            /*
             * Fixed position
             */
            if ( cur_ap->attr_length != 1 )
                offset += offset%2;   /* align the domain       */
            if ( typeof(cur_ap->attr_type) == Ktype_string )
            {
                /*
                 * Subsequent fields are variable position,
                 * but this field is still fixed.
                 * Length must be multiple of 2, since
                 * sensors write data in shorts.
                 */
                slen = strlen(buffer+offset) +1;
                cur_ap->attr_length = (slen%2)?
                    slen + 1  :  slen;
                cur_ap->attr_pos = offset;
                offset += cur_ap->attr_length;
                offset = -offset;       /* now it's variable pos */
            } /* if */
            else
            {
                cur_ap->attr_pos = offset;
                offset += cur_ap->attr_length;
            } /* else */
        } /* else */
    } /* foreach */
    if ( ((struct mon_cmd *)buffer)->length != offset>1 )
        ((struct mon_cmd *)buffer)->length = offset>1;
                                            /* reset length */
    } /* if */
} /* setposition */
/*
 *----------------------------------------------------------------------
 * getdomainbyname -
 *      return a pointer to the named domain in the tuple,
 *      NULL is returned if the domain isn't in the tuple.
 *----------------------------------------------------------------------
 */
attribute
getdomainbyname(rp,domname)
relation        rp;
char            *domname;
{
        /*
         * Search for the named domain, return pointer to domain
         */
        register SEQattribute   tmp_ap = NULL;      /* ptr for foreach */
        register attribute      cur_ap = NULL;      /* ptr for foreach */

        foreachinSEQattribute(rp->attributes, tmp_ap, cur_ap)
        {
                if ( strcmp(cur_ap->attr_name,domname) == 0 )
                        break;
        }
        if ( tmp_ap == NULL )                       /* not in attributes */
                return(NULL);
        else
                return(cur_ap);
} /* getdomainbyname */

/*
 *----------------------------------------------------------------------
 * rmdomainbyname -
 *      remove the named domain from the relation, if it is there
 *----------------------------------------------------------------------
 */
void
rmdomainbyname(rp,dname)
relation rp;                            /* relation containing domain */
char     *dname;                        /* name of domain             */
{
        attribute       ap;             /* pointer to attribute       */

        if ( (ap = getdomainbyname(rp, dname)) != NULL )
                removeSEQattribute(rp->attributes, ap);
} /* rmdomainbyname */

/*
 *----------------------------------------------------------------------
 * copyattribute -
 *      make a new copy of an attribute in memory
 *----------------------------------------------------------------------
 */
attribute
copyattribute(ap)
attribute       ap;                     /* attribute to be copied */
```

```
/*----------------------------------------------------------------------
 * tupleprint -
 *      print out in ascii the values for the tuple
 *      Label the values if label is true
 *----------------------------------------------------------------------
 */

int
tupleprint(fp, tp, label)
FILE    *fp;                            /* where to print           */
tuple   *tp;                            /* what to print            */
int     label;                          /* whether to label         */
{
        char         **format;          /* format to use for output */
        SEQattribute t_seq_ap;          /* tmp for loop             */
        attribute    ap;                /* tmp for loop             */


        if ( fp == NULL )
        {
            fprintf(stderr, "tupleprint: invalid file.\n");
            return(-1);
        }
        if ( tp == NULL || tp->relation == NULL
            || tp->relation->attributes == NULL )
        {
            fprintf(stderr, "tupleprint: invalid tuple.\n");
            return(-1);
        }

        /*
         * Select the appropriate printing format,
         * either labelled or unlabelled.
         */

        format = (label == PRINTLABELS)? l_format : u_format;

        /*
         * Print the relation name (if labelled) and each domain.
         */

        fprintf(fp, format[FMT_RELATION], tp->relation->rel_name);
        foreachinSEQattribute(tp->relation->attributes, t_seq_ap, ap)
        {
            switch(typeof(ap->attr_type))
            {
            case Ktype_string:
                fprintf(fp, format[FMT_STRING], ap->attr_name,
                    tp->record[ap->attr_pos]);
                break;
            case Ktype_boolean:
                fprintf(fp, format[FMT_BOOLEAN], ap->attr_name,
                    ((tp->record[ap->attr_pos] == 0) ? "F" : "T") );
                break;
            case Ktype_integer:
                switch(ap->attr_length)
                {
                case 1:
                    fprintf(fp, format[FMT_INT_1], ap->attr_name,
                        (int)tp->record[ap->attr_pos]);
                    break;
                case 2:
                    fprintf(fp, format[FMT_INT_2], ap->attr_name,
                        *(short *)&(tp->record[ap->attr_pos]));
                    break;
                case 4:
                    fprintf(fp, format[FMT_INT_4], ap->attr_name,
                        *(long *)&(tp->record[ap->attr_pos]));
                    break;
                default:
                    fprintf(fp, "UNKNOWN        ");
                }
                break;
            case Ktype_rational:
                fprintf(fp, format[FMT_RATIONAL], ap->attr_name,
                    *(float *)&(tp->record[ap->attr_pos]));
                break;
            default:
                fprintf(fp, "UNKNOWN    ");
            } /* switch */
        } /* foreach */
        (void)putc('\n', fp);                       /* end the line */
        if ( ferror(fp) )
        {
            perror("tupleprint");
            return(-1);
        }
        return(0);
} /* tupleprint */
```

---

readrecord.c

```
/* readrecord.c - readrecord */
```

```
/*
 * Copying consists of copying the name, the sensor id, and the
 * variable length flag, and then copying the key attributes (see
 * key_attr_tab in tuple_key.h) and the remaining attributes.
 */
if ( rp == NULL )                    /* don't have to do anything */
        return( NULL );

/*
 * Note that the name pointer is copied,
 * so that the space is shared.
 */

new_rp->rel_name       = rp->rel_name;
new_rp->rel_vlensensor = rp->rel_vlensensor;
new_rp->rel_sensor_id  = rp->rel_sensor_id;

/*
 * Must set up immutable (key) fields
 */

for ( i = 0; i < key_attr_tabsize; i++)
{
    ap = getdomainbyname(rp, key_attr_tab[i]);
    if ( ap == NULL )                    /* Should never happen! */
    {
        fprintf(stderr, "Copyrelation: error in keys\n");
        exit(1);
    }
    appendrearSEQattribute(new_rp->attributes, ap);
}
if ( emptySEQattribute(rp->attributes) )
    return(new_rp);             /* don't need to copy other attrs */

/*
 * Copy the attributes (mutable domains)
 */

foreachinSEQattribute(rp->attributes, tmp_seq_ap, ap)
{
    if ( inSEQattribute(new_rp->attributes, ap) )
        continue;                                /* don't copy keys */
    if ( (new_ap = copyattribute(ap)) == NULL )  /* didn't work */
    {
        fprintf(stderr,"copyrelation: couldn't copy attributes\n");
        return(NULL);
    }
    appendrearSEQattribute(new_rp->attributes, new_ap);
}

return(new_rp);
} /* copyrelation */

/*
 *-----------------------------------------------------------------------
 * copyschema -
 *      copy a schema.  This creates a completely separate copy
 *      of the schema so that changes to it will not affect the
 *      original schema.
 *-----------------------------------------------------------------------
 */
database
copyschema(schema)
database        schema;                         /* schema to be copied    */
{
        database    newschema = Ndatabase;      /* target for copy        */
        relation    rp,                         /* current pos in SEQrel. */
                    new_rp;                     /* copy of rp             */
        register
        SEQrelation tmp_seq_rp;                 /* for use in foreach     */

        /*
         * Copying consists of copying the name and
         * copying the sequence of relations.
         */

        if ( schema == NULL )               /* don't have to do anything */
            return( NULL );

        /*
         * Note that name is shared by both schemas.
         */

        newschema->database_name = schema->database_name;

        if ( emptySEQrelation(schema->relations) )
            return(newschema);                    /* don't have to copy */

        foreachinSEQrelation(schema->relations, tmp_seq_rp, rp)
        {
            if ( (new_rp = copyrelation(rp)) == NULL )   /* didn't work */
            {
                fprintf(stderr,"copyschema: couldn't copy relations\n");
                return(NULL);
            }
            appendrearSEQrelation(newschema->relations, new_rp);
        }
        return(newschema);

} /* copyschema */

/*
```

MINIKERNEL

README

```
This directory contains a set of files for emulating kernel calls.  When
it is compiled with "accountant" or "acct" of the Monitor system, it can
be used to test those programs without endangering the system.
        kern.c                  vehicle for kern sensors
        makefile                permits "make acct", make "accountant"
        minikern.c              kernel routines, call kern sys and ufs
        sys.c                   vehicle for ufs sensors
        ufs.c                   vehicle for ufs sensors
Other files that are used that are not in this directory:
        local_syscalls.c        syslocal and monitor system calls.
        /$(SOFTLAB)/include/monitor
                                headers for Monitor system
        /$(SOFTLAB)/src/monitoringkernel/monacct/acct.c
                                minimal accountant program
        /$(SOFTLAB)/src/accountant/
                                real accountant.
```

makefile

```
ACCOUNTANT=/usr/softlab/src/accountant
ACCT=/usr/softlab/src/monitoringkernel/monacct
CFILES=/usr/softlab/src/monitoringkernel/monsys/local_syscalls.c \
        minikern.c ufs.c kern.c sys.c

accountant:     $(ACCOUNTANT)/*.c $(CFILES)
        make -f $(ACCOUNTANT)/Makefile testaccountant
        mv testaccountant accountant


acct:   $(ACCT)/acct.c $(CFILES)
        make -f $(ACCT)/Makefile miniacct
        mv miniacct acct
```

minikern.c

```
/* minikern.c - copyin copyout syscall panic sleep */
#ifndef lint
static char rcsheader[] = "$Header";
#endif lint

#undef sun
#include <sys/param.h>
#include <sys/dir.h>
#include <sys/systm.h>
#include <sys/user.h>
#include <sys/proc.h>
#include <sys/syslocal.h>
#include <sys/stat.h>
#include <stdio.h>

/*
 *-----------------------------------------------------------------
 * This module emulates the three kernel calls necessary to permit
 * local_syscalls and the accountant work without actually going
 * through the kernel.
 *-----------------------------------------------------------------
 */

struct proc u_procp;
struct user u;                          /* kernel user structure */

/*
 *-----------------------------------------------------------------
 * copyin -
 *      copies a user buffer of a specified length to a kernel buffer
 *-----------------------------------------------------------------
 */
int
copyin(outbuf, inbuf, len)
register caddr_t inbuf,
                 outbuf;
register int     len;
{
        for (;len > 0; len--)
                *inbuf++ = *outbuf++;
```

```
#include <stdio.h>
#include <monitor/montypes.h>
#define FIXEDLEN sizeof(struct mon_cmd)*2

/*-------------------------------------------------------------------------
-
 * readrecord
 *      - reads an event record from a file pointer into a putevent struct
 *      - puts event record into structure pointed to by recd
 *      - returns the length of the record or -1 on an io error.
 *-------------------------------------------------------------------------
-
 */
readrecord(fp,recd)
FILE     *fp;
mon_putevent *recd;
{
        int      stat;                  /* for io calls */
        int      varlen;                /* length of record past cmd */
        short   *record=(short *)recd;  /* a little short hand */

        /*
         * Find length of record
         */

        if ( (stat = fread (record, sizeof(short), FIXEDLEN, fp)
             ) != FIXEDLEN
           )
                if (feof(fp))  return (0);      /* all done */
                else  return(-1);               /* some error */

        /*
         * Determine length in shorts remaining, (note recd==record)
         */

        varlen = (int)recd->cmd.length - FIXEDLEN;

        /*
         * Get remainder of record
         */

        if ( (stat = fread ((record + FIXEDLEN),
                            sizeof(short), varlen, fp)
             ) != varlen )
                return (-1);            /* didn't read whole record */

        return (recd->cmd.length);      /* return length in short ints */
}
```


writerecord.c

```
/* writerecord.c - writerecord */

#include <stdio.h>
#include <monitor/montypes.h>

/*-------------------------------------------------------------------------
-
 * writerecord
 *      - writes an event record from a buffer that starts with a mon_cmd
 *      - struct (a mon_putevent recd ) for the length in specified in
 *      - that struct.  returns what fwrite returns
 *-------------------------------------------------------------------------
-
 */
writerecord(fp,recd)
FILE     *fp;
mon_putevent *recd;
{
        short   *record=(short *)recd;  /* a little short hand */

        return( fwrite(record, sizeof(short), recd->cmd.length, fp) );
}
```

```
kern()
{
        struct inode In;
        struct file F, *fp;
        struct  {
                int tv_usec;
                int tv_sec;
                } time;
        struct a {
                int p_pid;
                }p;
        struct {
                struct a *u_procp;
                } u;

        In.i_dev = 2;
        In.i_number = 1;
        In.i_size = 14;
        fp = &F;
        fp->f_data = (caddr_t)&In;
#ifdef DEBUG
        fprintf(stderr,"In kern sensor\n");
        fprintf(stderr,"Sen_commands: addr1 = %d, addr2 = %d, [0] = X%x, [0]&6
4 = %d\n",
                sen_commands,
                &sen_commands[0],
                sen_commands[0],
                sen_commands[0]&64);
#endif
        time.tv_usec = 1500;
        time.tv_sec = 1500;
        p.p_pid = 12345;
        u.u_procp = &p;
        FileClose(((struct inode *)(fp->f_data))->i_dev,((struct inode *)(fp->
f_data))->i_number,((struct inode *)(fp->f_data))->i_size);
}
```

sys.c

```
#include "../h/types.h"
#include "../h/sys_sensors.h"
#include <stdio.h>
sys()
{
        struct  {
                int tv_usec;
                int tv_sec;
                } time;
        struct a {
                int p_pid;
                }p;
        struct {
                struct a *u_procp;
                } u;

#ifdef DEBUG
        fprintf(stderr, "In sys sensor\n");
        fprintf(stderr, "Sen_commands[0]: X%4x\n",
                sen_commands[0]);
#endif
        time.tv_usec = 1500;
        time.tv_sec = 1500;
        p.p_pid = 12345;
        u.u_procp = &p;
        ReadSensor(13,10,512,128);
        WriteSensor(13,10,512,128);
}
```

ufs.c

```
#include "../h/types.h"
#include "../h/ufs_sensors.h"
#include <stdio.h>
ufs()
{
        struct  {
                int tv_usec;
                int tv_sec;
                } time;
        struct a {
                int p_pid;
                }p;
        struct {
                struct a *u_procp;
                } u;
        short cname[12];
```

```
        return(len);
}

/*
 *----------------------------------------------------------------------
 * copyout -
 *      copies a kernel buffer of a specified length to a user buffer
 *----------------------------------------------------------------------
 */
int
copyout(inbuf, outbuf, len)
register caddr_t inbuf,
                 outbuf;
register int     len;
{
        for (;len > 0; len--)
                *outbuf++ = *inbuf++;
        return(len);
}

/*
 *----------------------------------------------------------------------
 * syscall -
 *      handle system calls.  If the call goes to SYSLOCAL,
 *      intercept it and call syslocal. Otherwise, fail.
 *----------------------------------------------------------------------
 */
int
syscall(d1,d2,u_buffer)
int     d1,d2;
register unsigned char *u_buffer;
{
        struct a {
                int     callno;
                unsigned char   *arg;
        } uap;
        char    dummy[8024];
        int     rcode;

        kern();                         /* generate kernel sensor data */
        sys();
        ufs();
        /*
         * Set up global user struct
         */
        u_procp.p_pid = 15;
        u.u_procp = &u_procp;
        if ( d1 == SYSLOCAL )
        {
                uap.callno = SYSL_MONITOR;
                uap.arg = u_buffer;
                u.u_ap = (int *)&uap;
                syslocal(SYSL_MONITOR,u_buffer);
                rcode = (u.u_error < 0) ? u.u_error : u.u_r.r_val1;
                return(rcode);
        }
        else
                return(-1);
}
/*
 *----------------------------------------------------------------------
 * panic
 *      print the argument and exit with 1
 *----------------------------------------------------------------------
 */

panic(msg)
char    *msg;                   /* NULL terminated string */
{
        fprintf(stderr, msg);
        fputc(stderr, '\n');    /* End with new line */
        exit(1);
}
/*
 *----------------------------------------------------------------------
 * sleep -
 *      simulates sleep system call
 *----------------------------------------------------------------------
 */

sleep(secs)
int     secs;
{
        ;                       /* Don't do anything */
}
```

---

kern.c

```
#include "../h/types.h"
#include "../h/inode.h"
#include "../h/file.h"
#include "../h/kern_sensors.h"
#include <stdio.h>
```

DISTRIBUTION/{MAKEFILE,SHORTALIGN.C}

Makefile

```
# Name of target kernel, used as basis for new one
KERNEL=GENERIC
# Name for new kernel
MONITOR=MONITOR
# Define for ifdefs
MONDEF=MONITOR
# Define for fetch dependencies, determined within
SHORTALIGN=
# Name of directory for monitor include files
MONINCLUDE=monitor
# Name of directory for Monitor system
MONSYS=monsys
# Location of system files
SYS=/sys

# Files that get patched
PATCH=kern_descrip.c ufs_syscalls.c ufs_nami.c ufs_alloc.c \
        sys_generic.c syscalls.c init_sysent.c
# C files in kernel
TARGETS=local_syscalls.c ${PATCH}
# Files that depend on ${MONINCLUDE}
INCLDEPS= h/ufs_sensors.h h/sys_sensors.h h/kern_sensors.h h/syslocal.h\
        monsys/local_syscalls.c monsys/acct.c monsys/miniacct.c \
        monsys/ufs.c monsys/sys.c monsys/kern.c monsys/acct_sensors.h\
        monsys/blindprint.c monsys/readrecord.c monsys/printevents.c \
        monsys/dumprecord.c monsys/shutdownacct.c \
        monitor/montypes.h monitor/mondefs.h monitor/monops.h \
        monitor/monerrcds.h

# Determine if this machine can only fetch shorts on short boundaries
config:
        cc -o shortalign shortalign.c
        /bin/rm -f makefile
        sed '/^SHORTALIGN/ s/=.*/='`shortalign`'/' Makefile > makefile
        @echo ">Make: Now you can make install"

# Put all the directories in place and substitute in for defines
install:        PATCH ${INCLDEPS}
        @echo ">Make: Installing system"
# Don't run sed unless you have to
        -grep -s ${MONINCLUDE} monsys/makefile || \
                sed '/^MONDEF/ s/MONITOR/${MONDEF}/' monsys/Makefile |\
                sed '/^MONINCLUDE/ s/monitor/${MONINCLUDE}/' > monsys/makefile
# Only do the moves and stuff once
        -test -f ${SYS}/h/ufs_sensors.h || cp h/ufs_sensors.h  ${SYS}/h
        -test -f ${SYS}/h/sys_sensors.h || cp h/sys_sensors.h  ${SYS}/h
        -test -f ${SYS}/h/kern_sensors.h || cp h/kern_sensors.h  ${SYS}/h
        -test -f ${SYS}/h/syslocal.h || cp h/syslocal.h  ${SYS}/h
        -test -d ${SYS}/${MONINCLUDE} ||  mkdir ${SYS}/${MONINCLUDE) && \
                cp monitor/* ${SYS}/${MONINCLUDE}
        -test -d /usr/include/${MONINCLUDE) || \
                (cd /usr/include; ln -s ${SYS}/${MONINCLUDE} ${MONINCLUDE})
        -test -f ${SYS}/${MONINCLUDE}/*.bak && /bin/rm -f ${SYS}/${MONINCLUDE)
/*.bak
        @echo ">Make: Sensors installed"
        -test -d ${SYS}/${MONSYS) || mkdir ${SYS}/${MONSYS) && \
                cp monsys/* ${SYS}/${MONSYS}
        -test -f ${SYS}/${MONSYS}/*.bak &&  /bin/rm -f ${SYS}/${MONSYS}/*.bak
        @echo ">Make: Making Accountant"
# Note that if this is run twice, nothing really happens
        (cd ${SYS}/${MONSYS}; make acct;  make blindprint; make shutdownacct)
        @echo ">Make: Installation done, run make new or make modify"


# Create a brand new kernel - most reliable way to go
new:    ${PATCH}
        @echo ">Make: creating new system"
# Assume that if this is done twice, that we should overwrite
        awk '{if ($$1 == "ident"){printf "ident\t\t\"${MONITOR}\"\n"} else pri
nt $$0}' ${SYS}/conf/${KERNEL) |\
        awk '{print $$0} /^options/ {if (T++ == 0){printf "options\t\t${MONDEF
} ${SHORTALIGN}\n"}}' > ${SYS}/conf/${MONITOR}
        /bin/rm -f  ${SYS}/conf/files.${MONITOR}
# This only makes or undates file.${MONITOR) if necessary
        -(test -f ${SYS}/conf/files.${MONITOR) && \
                grep -s "${MONSYS}" ${SYS}/conf/files.${MONITOR) ) || \
                sed 's/monsys/${MONSYS}/' files.MONITOR \
                > ${SYS}/conf/files.${MONITOR}
        -(cd ${SYS}/conf; \
        (test -d ../${MONITOR) || mkdir ../${MONITOR});\
        config ${MONITOR})
        (cd ${SYS}/${MONITOR}; make depend > depend.out 2>&1; \
                make vmunix > make.out 2>&1)
        @echo ">Make: New system all done"

# Modify an existing kernel -- trouble on some machines (SUN, e.g.)
# Unsupported, should do nearly the same as 'new', except should
# touch all .o files
modify: ${TARGETS} ${PATCH)
        @echo ">Make: Modifying" ${KERNEL}
        cp sedfile ${SYS}/${KERNEL)
        (cd ${SYS}/${KERNEL);   \
```

```
        char *compname=(char *)(cname);

#ifdef DEBUG
        fprintf(stderr, "In ufs sensor\n");
        fprintf(stderr, "Sen_commands: addr1 = %d, addr2 = %d, [0] = X%4x\n",
                sen_commands,
                &sen_commands[0],
                sen_commands[0]);
        fflush(stderr);
#endif
        time.tv_usec = 1500;
        time.tv_sec = 1500;
        p.p_pid = 12345;
        u.u_procp = &p;
        OpenSuccessful(1, 1024);
        NameStart(13,2048);
        strcpy(++compname,"seventy");
        NextComponent(13,2048,compname);
        strcpy(compname,"four");
        NextComponent(13,2048,compname);
        INodeCreate(13,2048);
        INodeDelete(13,2048);
}
```

SYS/H

README

These are the include files for the local system call and the kernel sensors.

kern_sensors.h	sensors for kern_* files
sys_sensors.h	sensors for sys_* files
syslocal.h	definitions for local system calls
ufs_sensors.h	sensors for ufs_* files

kern_sensors.h

```
/* Sensor macros for the Kern process.
   Generated from kernel.sen on June 14, 1984 (by hand).
   Contains the following macros:
        FileClose(device,inumber,finalsize)
*/
#ifdef KERNEL
#include "../monitor/mondefs.h"
#include "../monitor/montypes.h"
#ifndef ntohs
#include "../netinet/in.h"
#endif
#else
#include <monitor/mondefs.h>
#include <monitor/montypes.h>
#ifndef ntohs
#include <netinet/in.h>
#endif
#endif
#define Timestamp (int)((time.tv_sec << 15) | (time.tv_usec >> 5))

extern int      mon_semaphore;
extern unsigned char *mon_write_ptr;
extern unsigned char *mon_eventvector_end;
extern int   mon_eventvector_count;
extern int   mon_oflow_count;
extern unsigned short mon_enablevector[];
extern unsigned char *Wraparound();

#ifdef MONITOR
#define FileClose(device,inumber,finalsize)                                  \
if (*(mon_enablevector+0) & (1<<6))                                          \
    {                                                                        \
    if (mon_semaphore++ == 0)                                                \
        {                                                                    \
        if (mon_eventvector_count <                                          \
                MON_EVENTVECSIZE - 14*2 - sizeof(mon_errrec))                \
            {                                                                \
            register mon_putevent *reg_ptr = (mon_putevent *)mon_write_ptr;\
            register short    *sen_fields = reg_ptr->fields;                 \
            mon_printf(("FileClose: mon_write_ptr = %d,\t", mon_write_ptr));\
            reg_ptr->cmd.type    = MONOP_PUTEVENT_INT;                       \
            reg_ptr->cmd.length  = sen_fields+2 - (short *)reg_ptr;          \
            mon_eventvector_count += reg_ptr->cmd.length*2;                  \
            reg_ptr->eventnumber = (short)6;                                 \
            reg_ptr->performer   = 0;                                        \
            reg_ptr->object      = ( (short)device<<16 ) |                   \
                                 ( (short)inumber & 0xffff);                 \
            reg_ptr->initiator   = u.u_procp->p_pid;                         \
            reg_ptr->timestamp   = (int)Timestamp;                           \
            *(int *)(sen_fields) = (int)finalsize;                           \
            sen_fields          += 2;                                        \
            if ( sen_fields > (short *)mon_eventvector_end )                 \
                mon_write_ptr = Wraparound((unsigned char *)sen_fields);\
            else                                                             \
                mon_write_ptr = (unsigned char *)sen_fields;                 \
            mon_printf(("%d\n", mon_write_ptr));                             \
            }                                                                \
        else mon_oflow_count++;                                              \
        mon_semaphore--;                                                     \
        }                                                                    \
    }                       /* end FileClose */
#else
#define FileClose(device,inumber,finalsize)
#endif
```

```
	mv makefile makefile.${KERNEL}; \
	mv makefile.bak makefile.bak.${KERNEL}; \
	sed -f sedfile makefile.bak.${KERNEL} > makefile; \
	make depend > depend.out 2>&1; \
	make lint > lint.out 2>&1; \
	make vmunix > make.out 2>&1)
	@echo ">Make: all done"

cleanmodify:
	(cd ${SYS}/${KERNEL}; \
	mv makefile.${KERNEL} makefile; \
	mv makefile.bak.${KERNEL} makefile.bak)

PATCH:
	(cd Patch; make patch; mv patch ..)

${INCLDEPS}::
# Only apply changes if necesary
# Sorry about two negatives being required.
# Since bak is the original, we don't want to overwrite it
	-test monitor = ${MONINCLUDE} || \
		egrep -s '^#incl.*[</]${MONINCLUDE}' $@ || \
		(test -f $@.bak || mv $@ $@.bak); \
		sed '/^#include.*[<\/]monitor\// s/monitor/${MONINCLUDE}/' $@.
bak | \
	-	sed '/^#ifdef[  ]MONITOR/ s/MONITOR/${MONDEF}/' > $@)

# This is used to modify the kernel's makefile
${TARGETS}::	SEDFILE makefile
	echo 's,${SYS}/sys/$@,${SYS}/${MONSYS}/$@,g' >> sedfile
	echo 's/\/sys\/sys\/$@/\/sys\/${MONSYS}\/$@/g' >> sedfile


# Clear sedfile
SEDFILE:
	>sedfile

# This actually installs the sensors and system call support in source
${PATCH}::
	@echo ">Make: Applying patches"
	patch ${SYS}/sys/$@ ${SYS}/${MONSYS}/$@.patch -D ${MONDEF} -o ${SYS}/$
(MONSYS)/$@
```


shortalign.c

```
/* shortalign.c - main, sighandle */

#include <signal.h>
#include <stdio.h>

/*
 *----------------------------------------------------------------
 * shortalign
 *	- if the machine requires aligned fetches, output is
 *		SHORTALIGN
 *	- else nothing
 *----------------------------------------------------------------
 */

main()
{
	static short sarray[2]={0,0};	/* To insure alignment */
	char *charptr=(char *)sarray;
	short shold;
	void sighandle();

	/*
	 * Basic operation is to misalign charptr and then cast it
	 * as a short, should cause a bus error if there are troubles.
	 */

	signal(SIGBUS,sighandle);
	shold = *(short *)(charptr +1);
	printf("\n");

}
/*
 *----------------------------------------------------------------
 * sighandle
 *	- prints out 'SHORTALIGN' when called
 *----------------------------------------------------------------
 */
void
sighandle()
{
	printf(",SHORTALIGN\n");
	exit(0);
}
```

```
            mon_write_ptr = (unsigned char *)sen_fields;          \
        mon_printf(("%d\n", mon_write_ptr));                      \
        }                       /* end writesensor */             \
    else mon_oflow_count++;                                       \
    mon_semaphore--;                                              \
    }
}
#else
#define WriteSensor(device,inumber,filepos,actualcount)
#endif
```

---

ufs_sensors.h

```
/* Sensor macros for the UFS process.
   Generated from kernel.sen on June 14, 1984 (by hand).
   June 16,1985: added macros for char string handling for
        machines that can't fetch from arbitrary boundaries:
        PackStr(ptr)            - mondefs.h
        NotEOS(ptr,last)        - mondefs.h
        type mon_string         - montypes.h
        All of these are dependent on SHORTALIGN
   Contains the following macros:
        OpenSuccessful(mode, initsize)
        NameStart(device,inumber)
        NextComponent(device,inumber,filename)
        INodeCreate(device,inumber)
        INodeDelete(device,inumber)
*/
#ifdef KERNEL
#include "../monitor/mondefs.h"
#include "../monitor/montypes.h"
#ifndef ntohs
#include "../netinet/in.h"
#endif
#else
#include <monitor/mondefs.h>
#include <monitor/montypes.h>
#ifndef ntohs
#include <netinet/in.h>
#endif
#endif
#define Timestamp (int)((time.tv_sec << 15) | (time.tv_usec >> 5))
```



```
extern int      mon_semaphore;
extern unsigned char *mon_write_ptr;
extern unsigned char *mon_eventvector_end;
extern int   mon_eventvector_count;
extern int   mon_oflow_count;
extern short mon_enablevector[];
extern unsigned char *Wraparound();

/* The parameters for OpenSuccessful are:
        mode :          mode;
        initsize :      ip->di_size;
*/
#ifdef MONITOR
#define OpenSuccessful(mode,initsize)                                 \
if (mon_enablevector[0] & 1<<5)                                       \
    {                                                                 \
    if (mon_semaphore++ == 0)                                         \
        {                                                             \
        if (mon_eventvector_count <                                   \
                MON_EVENTVECSIZE - 13*2 - sizeof(mon_errrec))         \
            {                                                         \
            register mon_putevent *reg_ptr = (mon_putevent *)mon_write_ptr;\
            register short      *sen_fields = reg_ptr->fields;        \
            reg_ptr->cmd.type    = MONOP_PUTEVENT_INT;                \
            reg_ptr->cmd.length  = sen_fields+3-(short *)reg_ptr;     \
            mon_eventvector_count       += reg_ptr->cmd.length*2;     \
            reg_ptr->eventnumber         = (short)5;    /* id of sensor */\
            reg_ptr->performer   = 0;                                 \
            reg_ptr->object              = 0;           /* no object */ \
            reg_ptr->initiator   = u.u_procp->p_pid;                  \
            reg_ptr->timestamp   = (int)Timestamp;                    \
                                                /* type */            \
            *sen_fields++        = (short)mode;         /* int */     \
            *(int *)sen_fields   = (int)initsize;       /* dint */    \
            sen_fields          += 2;                                 \
            if ( sen_fields > (short *)mon_eventvector_end )          \
                mon_write_ptr = Wraparound((unsigned char *)sen_fields);\
            else                                                      \
                mon_write_ptr = (unsigned char *)sen_fields;          \
            }                                                         \
        else mon_oflow_count++;                                       \
        mon_semaphore--;                                              \
        }                                                             \
    }    /* end OpenSuccessful */
#else
#define OpenSuccessful(mode,initsize)
#endif

/* parameters for NameStart are from:
    device :   ip->i_dev
    inumber:   ip->i_number
```

sys_sensors.h

```
/* Sensor macros for the Sys process.
   Generated from kernel.sen on June 14, 1984 (by hand).
   Contains the following macros:
        ReadSensor(device,inumber,filepos,actualcount)
        WriteSensor(device,inumber,filepos,actualcount)
*/
#ifdef KERNEL
#include "../monitor/mondefs.h"
#include "../monitor/montypes.h"
#ifndef ntohs
#include "../netinet/in.h"
#endif
#else
#include <monitor/mondefs.h>
#include <monitor/montypes.h>
#ifndef ntohs
#include <netinet/in.h>
#endif
#endif
#define Timestamp (int)((time.tv_sec << 15) | (time.tv_usec >> 5))

extern int      mon_semaphore;
extern unsigned char *mon_write_ptr;
extern unsigned char *mon_eventvector_end;
extern int   mon_eventvector_count;
extern int   mon_oflow_count;
extern short mon_enablevector[];
extern unsigned char *Wraparound();

#ifdef MONITOR
#define ReadSensor(device,inumber,filepos,actualcount)                  \
if ( *(mon_enablevector+0) & 1<<8 )                                     \
    {                                                                   \
    if (mon_semaphore++ == 0)                                           \
        {                                                               \
        if (mon_eventvector_count <                                     \
            MON_EVENTVECSIZE - 15*2 - sizeof(mon_errrec))               \
            {                                                           \
            register mon_putevent *reg_ptr = (mon_putevent *)mon_write_ptr;\
            register short    *sen_fields = reg_ptr->fields;            \
            mon_printf(("ReadSensor: mon_write_ptr = %d,\t", mon_write_ptr));\
            reg_ptr->cmd.type     = MONOP_PUTEVENT_INT;                 \
            reg_ptr->cmd.length   = sen_fields+3 - (short *)reg_ptr;    \
            mon_eventvector_count += reg_ptr->cmd.length*2;             \
            reg_ptr->performer    = 0;                                  \
            reg_ptr->eventnumber = 8;                                   \
            reg_ptr->object       = ((short)device<<16 )|               \
                                    ((short)inumber  & 0xffff);         \
            reg_ptr->initiator    = u.u_procp->p_pid;                   \
            reg_ptr->timestamp    = (int)Timestamp;                     \
            *(int *)sen_fields    = (int)filepos;                       \
            sen_fields           += 2;                                  \
            *sen_fields++             = (short)actualcount;             \
            if ( sen_fields > (short *)mon_eventvector_end )            \
                mon_write_ptr = Wraparound((unsigned char *)sen_fields);\
            else                                                        \
                mon_write_ptr = (unsigned char *)sen_fields;            \
            mon_printf(("%d\n", mon_write_ptr));                        \
            }                                                           \
        else mon_oflow_count++;                                         \
        mon_semaphore--;                                                \
        }                                                               \
    }    /* end readsensor */
#else
#define ReadSensor(a,b,c,d)
#endif

#ifdef MONITOR
#define WriteSensor(device,inumber,filepos,actualcount)                 \
if (*(mon_enablevector+0) & 1<<9)                                       \
   {                                                                    \
    if (mon_semaphore++ == 0)                                           \
        {                                                               \
        if (mon_eventvector_count <                                     \
            MON_EVENTVECSIZE - 15*2 - sizeof(mon_errrec))               \
            {                                                           \
            register mon_putevent *reg_ptr = (mon_putevent *)mon_write_ptr;\
            register short    *sen_fields = reg_ptr->fields;            \
            mon_printf(("WriteSensor: mon_write_ptr = %d,\t", mon_write_ptr));
\
            reg_ptr->cmd.type     = MONOP_PUTEVENT_INT;                 \
            reg_ptr->cmd.length   = sen_fields+3 - (short *)reg_ptr;    \
            mon_eventvector_count += reg_ptr->cmd.length*2;             \
            reg_ptr->performer    = 0;                                  \
            reg_ptr->eventnumber = 9;                                   \
            reg_ptr->object       = ((short)device <<16 )|              \
                                    ((short)inumber  & 0xffff);         \
            reg_ptr->initiator    = u.u_procp->p_pid;                   \
            reg_ptr->timestamp    = (int)Timestamp;                     \
            *(int *)sen_fields    = (int)filepos;                       \
            sen_fields           += 2;                                  \
            *sen_fields++         = (short)actualcount;                 \
            if ( sen_fields > (short *)mon_eventvector_end )            \
                mon_write_ptr = Wraparound((unsigned char *)sen_fields);\
            else                                                        \
```

```
            reg_ptr->initiator   = u.u_procp->p_pid;                    \
            reg_ptr->timestamp   = (int)Timestamp;                      \
                                        /*type */                       \
            if ( sen_fields > (short *)mon_eventvector_end )            \
                mon_write_ptr = Wraparound((unsigned char *)sen_fields);\
            else                                                        \
                mon_write_ptr = (unsigned char *)sen_fields;            \
            }                                                           \
        else mon_oflow_count++;                                         \
        mon_semaphore--;                                                \
        }                                                               \
    }    /* end INodeCreate */
#else
#define INodeCreate(device,inumber)
#endif

/* parameters for INodeDelete are from:
    device :   ip->i_dev
    inumber:   ip->i_number
*/
#ifdef MONITOR
#define INodeDelete(device,inumber)                                     \
if (*(mon_enablevector+0) & 1<<7)                                       \
    {                                                                   \
    if (mon_semaphore++ == 0)                                           \
        {                                                               \
        if (mon_eventvector_count <                                     \
                MON_EVENTVECSIZE - 12*2 - sizeof(mon_errrec))           \
            {                                                           \
            register mon_putevent *reg_ptr = (mon_putevent *)mon_write_ptr;\
            register short      *sen_fields = reg_ptr->fields;          \
            reg_ptr->cmd.type    = MONOP_PUTEVENT_INT;                  \
            reg_ptr->cmd.length  = sen_fields - (short *)reg_ptr;       \
            mon_eventvector_count += reg_ptr->cmd.length*2;             \
            reg_ptr->eventnumber = (short)7;    /* id of sensor */      \
            reg_ptr->performer   = 0;                                   \
            reg_ptr->object      = ((short)device<<16) |    /* int */   \
                                   ((short)inumber&0xffff);   /* int */ \
            reg_ptr->initiator   = u.u_procp->p_pid;                    \
            reg_ptr->timestamp   = (int)Timestamp;                      \
                                                                        \
            if ( sen_fields > (short *)mon_eventvector_end )            \
                mon_write_ptr = Wraparound((unsigned char *)sen_fields);\
            else                                                        \
                mon_write_ptr = (unsigned char *)sen_fields;            \
            }                                                           \
        else mon_oflow_count++;                                         \
        mon_semaphore--;                                                \
        }                                                               \
    }    /* end INodeDelete */
#else
#define INodeDelete(device,inumber)
#endif
```


---

syslocal.h

```
/* rcsid = $Header: syslocal.h,v 1.3 85/11/12 21:33:10 duncans Exp $ */
#define SYSLOCAL        151     /* syslocal system call index */
#define SYSL_NARGS      5       /* Max number of args allowed to */
                                /* local system calls. */

/* local system calls */
#define SYSL_MONITOR    1
```

```
*/
#ifdef MONITOR
#define NameStart(device,inumber)                                       \
if (mon_enablevector[0] & 1<<1)                                         \
    {                                                                   \
    if (mon_semaphore++ == 0)                                           \
        {                                                               \
        if (mon_eventvector_count <                                     \
                MON_EVENTVECSIZE - 12*2 - sizeof(mon_errrec))           \
            {                                                           \
            register mon_putevent *reg_ptr = (mon_putevent *)mon_write_ptr;\
            register short      *sen_fields = reg_ptr->fields;          \
            reg_ptr->cmd.type    = MONOP_PUTEVENT_INT;                  \
            reg_ptr->cmd.length  = sen_fields - (short *)reg_ptr;       \
            mon_eventvector_count += reg_ptr->cmd.length*2;             \
            reg_ptr->eventnumber = (short)1;    /* id of sensor */      \
            reg_ptr->performer   = 0;                                   \
            reg_ptr->object      = ((short)device<<16) |    /* int */   \
                                   ((short)inumber&0xffff); /* int */   \
            reg_ptr->initiator   = u.u_procp->p_pid;                    \
            reg_ptr->timestamp   = Timestamp;                           \
            if ( sen_fields > (short *)mon_eventvector_end )            \
                mon_write_ptr = Wraparound((unsigned char *)sen_fields);\
            else                                                        \
                mon_write_ptr = (unsigned char *)sen_fields;            \
            }                                                           \
        else mon_oflow_count++;                                         \
        mon_semaphore--;                                                \
        }                                                               \
    }    /* end NameStart      */
#else
#define NameStart(device,inumber)
#endif

/* parameters for NextComponent are from:
    device :   ip->i_dev
    inumber:   ip->i_number
    filename:  u.u_dbuf[0..15]
*/
#ifdef MONITOR
#define NextComponent(device,inumber,filename)                          \
if (*(mon_enablevector+0) & 1<<2)                                       \
    {                                                                   \
    if (mon_semaphore++ == 0)                                           \
        {                                                               \
        if (mon_eventvector_count <                                     \
                MON_EVENTVECSIZE - 260 - sizeof(mon_errrec))            \
            {                                                           \
            register mon_putevent *reg_ptr  = (mon_putevent *)mon_write_ptr;\
            register short      *sen_fields  = reg_ptr->fields;         \
            register mon_string sen_f_ptr = (mon_string )filename;      \
            register mon_string sen_f_end = (sen_f_ptr+127*2/sizeof(mon_string
));\
            register short sen_length;                                  \
            reg_ptr->eventnumber = (short)2;    /* id of sensor */      \
            reg_ptr->performer   = 0;                                   \
            reg_ptr->object      = ((short)device<<16) |    /* int */   \
                                   ((short)inumber&0xffff);   /* int */ \
            reg_ptr->initiator   = u.u_procp->p_pid;                    \
            reg_ptr->timestamp   = (int)0;      /* nil timestamp */     \
            do { *sen_fields++   = PackStr(sen_f_ptr); }                \
                while (NotEOS(sen_f_ptr,sen_f_end));                    \
            *(sen_fields - 1)   &= ntohs(0xff00);                       \
            sen_length           = sen_fields - (short *)reg_ptr;       \
            reg_ptr->cmd.type    = MONOP_PUTEVENT_INT;                  \
            reg_ptr->cmd.length  = sen_length;                          \
            mon_eventvector_count += sen_length*2;                      \
            if ( sen_fields > (short *)mon_eventvector_end )            \
                mon_write_ptr   = Wraparound((unsigned char *)sen_fields);\
            else                                                        \
                mon_write_ptr   = (unsigned char *)sen_fields;          \
            }     /* if still room in vector */                         \
        else mon_oflow_count++;                                         \
        mon_semaphore--;                                                \
        }                                                               \
    }   /* end NextComponent  */
#else
#define NextComponent(device,inumber,filename)
#endif

/* parameters for INodeCreate are from:
    device :   ip->i_dev
    inumber:   ip->i_number
*/
#ifdef MONITOR
#define INodeCreate(device,inumber)                                     \
if (*(mon_enablevector+0) & 1<<3)                                       \
    {                                                                   \
    if (mon_semaphore++ == 0)                                           \
        {                                                               \
        if (mon_eventvector_count <                                     \
                MON_EVENTVECSIZE - 12*2 - sizeof(mon_errrec))           \
            {                                                           \
            register mon_putevent *reg_ptr = (mon_putevent *)mon_write_ptr;\
            register short      *sen_fields = reg_ptr->fields;          \
            reg_ptr->cmd.type    = MONOP_PUTEVENT_INT;                  \
            reg_ptr->cmd.length  = sen_fields - (short *)reg_ptr;       \
            mon_eventvector_count += reg_ptr->cmd.length*2;             \
            reg_ptr->eventnumber = (short)3;    /* id of sensor */      \
            reg_ptr->performer   = 0;                                   \
            reg_ptr->object      = ((short)device<<16) |    /* int */   \
                                   ((short)inumber&0xffff);   /* int */ \
```

```
struct mon_pevt
        {
        struct mon_cmd  cmd;
        short           eventnumber,
                        performer;
        long            object;
        short           initiator;
        long            timestamp;
        short           fields[EVENT_LIMIT];
        };
typedef struct mon_pevt mon_putevent;

struct mon_erec
        {
        struct mon_cmd  cmd;
        long            val;
        };
typedef struct mon_erec mon_errrec;

struct mon_gevt
        {
        struct mon_cmd  cmd;
        unsigned short  req_length;
        short           *acct_buf_ptr; /* This is a buffer in user's area */
        };
typedef struct mon_gevt mon_getevent;

struct mon_request
        {
        short       targetpid,
                    eventnumber,
                    enablevalue;
        };

struct mon_preq
        {
        struct mon_cmd     cmd;
        struct mon_request req;
        };
typedef struct mon_preq mon_putreq;
typedef struct mon_preq mon_getreq;

struct mon_command
        {
        union   {
                struct mon_cmd cmd; /* other cmds only have first 2 fields */
                mon_putevent   pevt;
                mon_getevent   gevt;
                mon_putreq     preq;
                } u_event;
        };

typedef struct mon_command mon_command;
```

mondefs.h

```
#ifdef MONITOR
/*
 * Declarations for the monitor system call.
 */
#include <monitor/monops.h>             /* operator defines and MONOPS macro *
/
#include <monitor/monerrcds.h>  /* error codes from syscall */

#define REQ_LENGTH 1024*50
#define ACCT_BUFFER 0x10be8
#ifdef MONDEBUG
#define MON_EVENTVECSIZE  35000
#define mon_printf(a) printf a  /* Monitor kernel debugging statements */
#else
#define MON_EVENTVECSIZE  50000 /* Size of vect that stores event recs */
#define mon_printf(a)
#endif
#define MON_EVENTRECSIZE sizeof (mon_command) /* Max size of event record */
#ifdef SHORTALIGN               /* For fetches that may need alignment */
#define PackStr(ptr) ( ntohs((*ptr<<8) | (*(ptr+1))) )
#define NotEOS(ptr,last) (ptr <= last && *ptr++&0xff && *ptr++&0xff)
#else
#define PackStr(ptr) ((short)*ptr)
#define NotEOS(ptr,last) (ptr <= last && *ptr&0x00ff && *ptr++&0xff00)
#endif
#endif MONITOR
```

monerrcds.h

```
#define MON_ALRDY_INIT  -1
#define MON_NOT_INIT    -2
#define MON_BUFF_FULL   -3
#define MON_REQ_NOT_FND -4
#define MON_NOT_ACCTNT  -5
#define MON_REQ_OFLOW   -6
#define MON_INV_CMD     -7
#define MON_SYS_ERR     -8
#define MON_CONCURRENCY_ERR -9
```

monops.h

```
#define MONOP_INIT              9
#define MONOP_PUTEVENT_INT      1
#define MONOP_PUTEVENT_EXT      6
#define MONOP_GETEVENTS         2
#define MONOP_PUTREQ            10
#define MONOP_GETREQ            11
#define MONOP_SHUTDOWN          12
#define MONOP_OFLOW             13
#define MONOP_NO_REQ            14


#define MONOP(o, l) ((short) (o | (l << 8)))
```

montypes.h

```
#define EVENT_LIMIT 256
#ifdef SHORTALIGN
typedef char *  mon_string;
#else
typedef short * mon_string;
#endif

struct  mon_cmd
        {
        char    type,
                length;
        };
```

```
    uap = (struct a *)u.u_ap;
    switch (uap->calino) {              /* each case should be #ifdef'd   */
#ifdef MONITOR
    case SYSL_MONITOR:
        monitor(uap->arg[0]);
        break;
#endif MONITOR

    default:
        u.u_error = EINVAL;
        break;
    }
    return;
}

/*
 *----------------------------------------------------------------------
 * monitor --
 *    The purpose of this system call is to allow communication
 *  between sensors in target programs and a monitoring process.
 *  Sensors send event records to and retrieve commands from
 *  monitor while the ACCOUNTANT sends commands and retrieves event
 *  records.
 *
 *  Written by Dave Doerner for the Monitor project (CS145) 5/2/83
 *  Modified by Stephen Duncan as part of MS project,
 *    Changed data buffer to a circular queue
 *    Changed to utilize structs in buffer
 *    Revamped much of the code: mnemonics, flow inside cases
 *----------------------------------------------------------------------
 */

static
monitor (buffer)                        /* SYSTEM CALL */
u_char   *buffer;                       /* Address of command */
{
#ifdef MONITOR
#include <sys/kernel.h>
#define CALLERID u.u_procp->p_pid       /* process id of caller         */

    mon_command  u_command;             /* Receives command             */
    mon_command *u_cmd_ptr = &u_command;

    u_char      *Wraparound();          /* handles ring buffer wraparound */

    int      i, j;                      /* loop temporaries             */
    int      cmd_length;                /* length of command in chars   */
    int      notzero,                   /* booleans                     */
             match;

    mon_printf (("*********SYSMON CALLED***********\n"));
```

```
    if (mon_semaphore)                  /* concurrency check            */
    {
        /*
         * A concurrency error has occurred.
         * - Turn off the kernel sensors.
         * - Return an error.
         * Note that until the error passes, no data can be read from
         *      the event vector, or added to it by PUTEVENTS.  This is
         *      to try and minimize problems with the pointers.  This
         *      gets cleared only when and if mon_semaphore is
         *      appropriately decremented.  If two kernel sensors
         *      caused the concurrency, it will never clear, only
         *      rebooting will help then.  Since this indicates buggy
         *      code, the code with sensors should be recompilied with
         *      the -DMON_ASSERT option.
         */
        for (i = 0; i < MON_ENABLEVECTORSIZE; i++)
            mon_enablevector[i] = 0;    /* turn off sensors             */
        u.u_r.r_val1 = MON_CONCURRENCY_ERR;
        return;
    }
    else mon_semaphore++;               /* begin critical section       */

    u.u_r.r_val1 = 0;                   /* return val is initially 0    */

    /*
     * Copy in command
     *  first copy in struct that starts command to determine length
     *  then copy in whole command overtop of the struct for that length
     *  Since the type of command isn't known yet, the union version
     *  of a command is used.  This prevents any alignment problems that
     *  might occur with the structs.
     */
    mon_printf (("buffer: 0x%x\n", (int) buffer));
    if ( copyin ((caddr_t)buffer, (caddr_t)u_cmd_ptr, sizeof(struct mon_cmd))
                                                                        )
    {
        u.u_error = EFAULT;
        u.u_r.r_val1 = MON_SYS_ERR;     /* signifies system error */
        mon_semaphore--;                /* exit critical section  */
        return;
    }

    mon_printf (("command = %d\n", u_cmd_ptr->u_event.cmd.type));
    cmd_length = (int)u_cmd_ptr->u_event.cmd.length * 2;
                                        /* copyin deals in chars */
    mon_printf (("length = %d\n", cmd_length));

    if (cmd_length > MON_EVENTRECSIZE)
    {
        u.u_error = EINVAL;             /* invalid argument to system call */
```

local_syscalls.c

```
static char rcsid[] = "$Header: local_syscalls.c,v 1.3 85/11/12 20:35:36 dunca
ns Exp $";

/* local_syscalls.c -- syslocal, monitor, Wraparound */

#include "../h/param.h"
#include "../h/dir.h"
#include "../h/systm.h"
#include "../h/user.h"
#include "../h/proc.h"
#include "../h/syslocal.h"
#include "../monitor/mondefs.h"
#include "../monitor/montypes.h"

/*
 *----------------------------------------------------------------------
 * This file should contain all system calls that use syslocal.
 * Each local routine should
 *      - be of type static to prevent interference with the
 *        rest of the system.
 *      - have its global variables and defines with a unique prefix
 *      - have #ifdefs to control its compilation in the system
 *----------------------------------------------------------------------
 */

#define FALSE           0
#define TRUE            1

#ifdef MONITOR
/*
 * Declarations for the monitor system call.
 */

#define MON_OFLOWRECSIZE        sizeof(mon_errrec) /* In chars          */
#define MON_ENABLEVECTORSIZE    12      /* Chars in enable vector       */
#define MON_REQLISTSIZE         256     /* No. of entries in request list */
#define MON_REQOPENSLOT         0       /* Marks open slot in req. list */
#define MON_SUPERUSERUID        0       /* uid of root                  */
#define MON_BADPID              -1      /* For marking accountant_pid   */

#ifdef  MON_ASSERT
#define mon_assert(a,b)         if (a) panic(b)
#else
#define mon_assert(a,b)
#endif

/*
 *  monitor global ring buffer variables,
 *  initialized at compile time and when
 *  sensors are turned OFF
 */

u_char  *mon_write_ptr,         /* Write pointer in mon_eventvector     */
        *mon_read_ptr,          /* Read pointer in mon_eventvector      */
        *mon_eventvector_end;   /* First pos after buffer, start of appx */
int     mon_eventvector_count;  /* No. of chars of valid event records in mon_
eventvector */
int     mon_semaphore = 0;      /* Used to detect concurrancy           */
int     mon_oflow_count = 0;    /* Event record overflow                */
u_short mon_enablevector[MON_ENABLEVECTORSIZE];
                                /* enable flags for sensors             */

/*
 * Local variables for monitor
 */

u_char  mon_eventvector[MON_EVENTVECSIZE + MON_EVENTRECSIZE];
                                        /* Event record ring buffer     */
mon_errrec      mon_oflowrec = {        /* Buffer full indicator        */
    {MONOP_OFLOW,MON_OFLOWRECSIZE/2},   /* struct mon_cmd               */
    0 };                                /* count                        */
mon_errrec      mon_noreq = {           /* Err rec for no req in queue  */
    {MONOP_NO_REQ, MON_OFLOWRECSIZE/2}, /* struct mon_cmd               */
    0 };                                /* pid                          */
int     mon_initflag = FALSE;           /* initialize one time only     */
int     mon_accountant_pid = MON_BADPID; /* identity of accountant      */
struct  mon_request mon_requests[MON_REQLISTSIZE]; /* request list for users *
/

#endif MONITOR

/*
 *----------------------------------------------------------------------
 * syslocal
 *      Perform local system call services
 *      This drives all other local system calls
 *----------------------------------------------------------------------
 */
syslocal() {
    register struct a {
        int callno;
        int arg[SYSL_NARGS];            /* SYSL_NARGS is max args allowed */
    } *uap;
```

```
        {
            /*
             * Still room in ring buffer
             *      copy in the event record, point to the next opening,
             *      handle the wraparound condition.
             */
            register mon_putevent* pevt_ptr = (mon_putevent *)mon_write_pt
r;

            *pevt_ptr       = *(mon_putevent *)u_cmd_ptr;
            mon_write_ptr  += cmd_length;
            if (mon_write_ptr >= mon_eventvector_end)
                mon_write_ptr = Wraparound(mon_write_ptr);
            mon_eventvector_count += cmd_length;
        }  /* if */
        else
        {
            /*
             * We ran out of room in the buffer --
             *      set a flag so that an error record will
             *      be put in at GETEVENTS
             */
            mon_oflow_count++;
            u.u_r.r_val1 = MON_BUFF_FULL;
            mon_printf (("OVERFLOW:   vecptr = %D \n",
                (mon_write_ptr - mon_eventvector)));
            break;
        }  /* else we overflowed*/

        mon_printf (("vecptr= %d \n", (mon_write_ptr - mon_eventvector)));
        u.u_r.r_val1 = 0;
        break;
    }  /* else we are initialized */

case MONOP_GETEVENTS:
/*
 * Read all event records in vector.
 * Copies event records into buffer specified by acct_buf_ptr.
 * Accountant only.
 * Returns number of chars written out when successful,
 *          MON_NOT_INIT if called before initialization
 *          MON_NOT_ACCT if called is not accountant or superuser
 * Handles writing of error records.
 * Must handle four cases:
 *          1) wraparound and event count > requested
 *          2) wraparound and event count > requested
 *          3) no wraparound and event count !> requested
 *          4) no wraparound and event count !> requested
 * The count vs request will be handled first,
 * then the presence or absence of wraparound.
 */
```



```
if (!(mon_initflag))            /* not initialized    */
{
    u.u_r.r_val1 = MON_NOT_INIT;
    mon_printf (("Not Initialized\n"));
    break;
}

else if ( !(CALLERID == mon_accountant_pid
        || u.u_uid == MON_SUPERUSERUID) )
{
    u.u_r.r_val1 = MON_NOT_ACCTNT;
    break;
}

else                            /* all ok, proceed */
{
    register mon_getevent *gevt = (mon_getevent *) u_cmd_ptr;
                                /* command is a get event    */
    register int      req_length = gevt->req_length * 2;
                                /* requested length in chars */
    register caddr_t  acct_buf_ptr  = (caddr_t)(gevt->acct_buf_ptr);
                                /* ptr to accountants buffer */
    register int      char_count;
                                /* char count to write out   */
    int               transfer_count;
                                /* counts chars to transfer  */

    mon_printf (("Read out of vector to Accountant\n"));

    if (mon_oflow_count != 0)   /* ran out of room before call */
    {
        /*
         * Copy in error record
         *      treat it just like an event record
         *      guaranteed room for it
         */
        register mon_errrec *w = (mon_errrec *)mon_write_ptr;

        *w = mon_oflowrec;
        w->val = mon_oflow_count;
        mon_write_ptr += sizeof(mon_errrec);
        mon_eventvector_count += sizeof(mon_errrec);
        if (mon_write_ptr >= mon_eventvector_end)
            mon_write_ptr = Wraparound(mon_write_ptr);
        if (mon_write_ptr >= mon_eventvector_end
            || mon_write_ptr < mon_eventvector)
            panic("monitor: GET_EVENTS: err recd: pointers invalid\n")

        mon_oflow_count = 0;
    }
```

```
    u.u_r.r_val1 = MON_SYS_ERR;     /* signifies system error          */
    mon_semaphore--;                /* exit critical section           */
    return;
}

if ( copyin((caddr_t)buffer, (caddr_t) u_cmd_ptr, cmd_length) )
{
    u.u_error = EFAULT;
    u.u_r.r_val1 = MON_SYS_ERR;     /* signifies system error */
    mon_semaphore--;                /* exit critical section  */
    return;
}
mon_printf (("length = %d\n", u_cmd_ptr->u_event.cmd.length));

mon_printf(("Right before switch: oflow = %d, noreqflag = %d\n",
    mon_oflow_count, mon_noreqflag));

/*
 * This switch is the driver of the system call.  Each case
 * corresponds to a command.  A pointer to a specific type
 * of command is cast to the generic command for each case.
 * The command structure and return values are command dependent.
 */
switch (u_cmd_ptr->u_event.cmd.type)
{
case MONOP_INIT:
/*
 * Initialization
 *  The request and event record vectors are initialized.
 *  Can only be called once before a shutdown, the caller
 *  becomes the accountant.  If called a second time,
 *  MON_ALRDY_INIT is returned.
 *  Normally returns the size of the event vector in chars.
 */
    if (mon_initflag)           /* must already be initialized    */
    {
        u.u_r.r_val1 = MON_ALRDY_INIT;
        mon_printf (("Already Initialized\n"));
        break;
    }
    /*
     * Set up pointers and counters for event vector
     */
    mon_write_ptr         = mon_eventvector;
    mon_read_ptr          = mon_eventvector;
    mon_eventvector_end   = &mon_eventvector[MON_EVENTVECSIZE];
    mon_eventvector_count = 0;
    mon_oflow_count       = 0;

    mon_initflag       = TRUE;      /* records whether initialized */
    mon_accountant_pid = CALLERID;  /* initializer is accountant   */
```



```
    mon_printf (("Accountant is %d\n", CALLERID));
    /*
     * Turn off kernel sensors, just in case
     */
    for (i = 0; i < MON_ENABLEVECTORSIZE; i++)
        mon_enablevector[i] = 0;
    /*
     * Initialize request vector to all entries open
     */
    for (i = 0; i < MON_REQLISTSIZE; i++)
        mon_requests[0].eventnumber = MON_REQOPENSLOT;
    mon_printf (("Initialization done: oflow = %d, noreqflag = %d\n",
            mon_oflow_count,
            mon_noreqflag));
    u.u_r.r_val1 = MON_EVENTVECSIZE;        /* return size of ring buffer
*/
    break;


case MONOP_PUTEVENT_INT:
case MONOP_PUTEVENT_EXT:
/*
 * Write event record in vector
 *  Performer and timestamp fields are filled in and
 *  the record is put into the event vector.
 *  Returns 0 if the put was successful,
 *          MON_NOT_INIT if before the initialization,
 *          MON_BUF_FULL if no room in the vector.
 */
    if (!(mon_initflag))            /* not initialized */
    {
        u.u_r.r_val1 = MON_NOT_INIT;
        mon_printf (("Not Initialized\n"));
        break;
    }
    else
    {
        register mon_putevent *pevt;

        /*
         * fill in certain fields of event record
         */
        pevt = (mon_putevent *) u_cmd_ptr;
        pevt->performer = (short) CALLERID; /* fill in pid */
        pevt->timestamp = (long) (time.tv_sec << 15 | time.tv_usec >> 5);
                                /* Generate time stamp */

        mon_printf (("Time stamp taken   time= %d\n", pevt->timestamp));
        if ((mon_eventvector_count + cmd_length < MON_EVENTVECSIZE - MON_O
FLOWRECSIZE)
            && !mon_oflow_count)
```

```
                mon_assert( (mon_eventvector_count < 0),
                    "monitor: GET_EVENT: wrap: vec count invalid\n");
                char_count   -= transfer_count;
                mon_assert( (char_count < 0),
                    "monitor: GET_EVENT: wrap: char_count invalid\n");
                u.u_r.r_vall += transfer_count;
                acct_buf_ptr  += transfer_count;    /* adjust pointer for
next copy */
            }
        }
        mon_printf(("GET_EVENT: &acctbuf= %d, char_count= %d, r_vall= %d\n
",
            acct_buf_ptr,
            char_count,
            u.u_r.r_vall));
        if ( char_count > 0)
        {
            transfer_count = char_count;
            if ( copyout ((caddr_t) mon_read_ptr,
                                    acct_buf_ptr,
                                    char_count) < 0)
                {
                    u.u_error    = EFAULT;
                    u.u_r.r_vall = MON_SYS_ERR;
                    break;
                }
            else
                {
                /*
                 * Adjust pointers and counts,
                 * do sanity check.
                 * Note that wraparound has already been handled.
                 */
                mon_read_ptr          += char_count;
                mon_eventvector_count -= char_count;
                u.u_r.r_vall          += char_count;
                                      /* add count from this copy */
                u.u_r.r_vall /= 2;    /* convert to shorts    */
                mon_assert( (mon_write_ptr > mon_eventvector_end
                    || mon_write_ptr < mon_eventvector
                    || mon_eventvector_count < 0
                    || char_count < 0),
                    "monitor: GET_EVENT: invalid pointers\n");
                }
        }
        else
        {
            u.u_r.r_vall = 0;               /* didn't do anything */
        }
    }
```



```
    break;                              /* only break in this case  */

case MONOP_PUTREQ:
/*
 * Write command
 *  This command is used by the accountant to enable
 *          kernel sensors and to put requests in to user
 *          programs.
 *  A request is stored in the request vector at the first
 *          available slot based on the targetpid.
 *          The accountant uses a 0 target pid to indicate
 *          that the kernel is the target.
 *  Returns 0 if successful
 *          MON_REQ_OFLOW if the request vector is full
 *          MON_NOT_INIT if called before initialization
 *          MON_NOT_ACCTNT if called not the accountant or
 *                  superuser.
 */
    if (!(mon_initflag))        /* not initialized    */
    {
        u.u_r.r_vall = MON_NOT_INIT;
        mon_printf (("Not Initialized\n"));
        break;
    }
    mon_printf (("PID of writer: %d\n",
        ((mon_putreq *)(u_cmd_ptr))->req.targetpid));
    /*
     * Only the accountant or the superuser are allowed
     */
    if (CALLERID == mon_accountant_pid || u.u_uid == MON_SUPERUSERUID)
    {
        register mon_putreq *preq = (mon_putreq *) u_cmd_ptr;

        if (preq->req.eventnumber <= 0)     /* validate eventnumber */
        {
            u.u_r.r_vall = MON_SYS_ERR;
            u.u_error = EINVAL;
            break;
        }
        mon_printf (("Enabling sensors\n"));
        if ( preq->req.targetpid == 0 )     /* in kernel */
        {
            short pos = preq->req.eventnumber;
            if ( preq->req.enablevalue )
                mon_enablevector[pos / 16] |= 1 << (pos % 16);
            else
                mon_enablevector[pos / 16] &= ~(1 << (pos % 16));
            mon_printf(("PUTREQ: enablevector: %d pos: %d enableval: %d",
                    mon_enablevector[0] & (1<<(pos%16)),
                    pos,
                    preq->req.enablevalue));
```

```
    /*
     * First, calculate the number of chars to write out.
     * This handles the cases relating to the size of the
     * accountant's buffer.
     */
            if (mon_eventvector_count > req_length)
            {
                /*
                 * Determine length of events that will fit in
                 * req_length integrally
                 */
                register u_char *p;                     /* ptr into vector
*/
                register int    accum_length = 0;       /* accumulated length
*/
                register int    short_count = req_length / 2;
                                                        /* decremented length
*/

                mon_printf(("GET_EVENT: ->length = %d, req_length = %d\n",
                        ((struct mon_cmd *)mon_read_ptr)->length,
                        short_count));

                /*
                 * while room left, reduce room and point to next recd
                 *      accumulate length in accum_length
                 *      remember to handle wraparound
                 */
                for (p = mon_read_ptr;
                     short_count > ((struct mon_cmd *)p)->length;
                     short_count -= ((struct mon_cmd *)p)->length
                    )
                {
                    accum_length += ((struct mon_cmd *)p)->length; /* shorts *
/
                    mon_printf(("GET_EVENT: p->len=%d, alen=%d, scount=%d \n",
                        ((struct mon_cmd *)p)->length,
                        accum_length,
                        short_count));
                    p += ((struct mon_cmd *)p)->length * 2;         /* u_chars
*/
                    if (p >= mon_eventvector_end)   p -= MON_EVENTVECSIZE;
                }
                char_count = accum_length * 2;

                /*
                 * char_count now has length of whole
                 * recds that will fit in request
                 */

            } /* if less requested than available */
```



```
            else /* more requested than available */
                char_count = mon_eventvector_count;

    /*
     * Time to copy out to the user area.
     * This handles the cases relating to wraparound.
     */
            /*
             * transfer_count is amount actually transferred in a
             * given invocation of copyout.
             * transfer_count is set to the size in chars of logical
             * older part of the ring buffer only when there is
             * wraparound physically starting at mon_read_ptr.
             */

            mon_printf(("IN GET_EVENT: wrap = %d, out = %d, req = %d\n",
                (mon_eventvector_end - mon_read_ptr),
                char_count,
                req_length));
            mon_printf(("Writer=%d reader=%d\n",
                (mon_write_ptr - mon_eventvector),
                (mon_read_ptr - mon_eventvector)));
            if (mon_write_ptr < mon_read_ptr)
            {
                mon_printf(("In Wrap.\n"));
                /*
                 * Write out all or portion of older part
                 */
                transfer_count =
                    ((mon_eventvector_end - mon_read_ptr) > char_count) ?
                    char_count  :  (mon_eventvector_end - mon_read_ptr);
                if (copyout ((caddr_t) mon_read_ptr,
                             (acct_buf_ptr),
                             transfer_count) < 0)
                {
                    u.u_error = EFAULT;
                    u.u_r.r_val1 = MON_SYS_ERR;
                    break;
                }
                else    /* copy succeeded */
                {
                    /*
                     * Adjust pointers and counts,
                     * do sanity checks
                     */
                    mon_assert( (transfer_count < 0),
                        "monitor: GET_EVENT: wrap value invalid");
                    mon_read_ptr   += transfer_count;
                    if (mon_read_ptr >= mon_eventvector_end)
                        mon_read_ptr -= MON_EVENTVECSIZE;
                    mon_eventvector_count    -= transfer_count;
```

```
     * Returns 0 when successful,
     *         MON_NOT_INIT if called before initialization,
     *         MON_NOT_ACCTNT if caller is not the accountant or
     *                 the superuser
     */
        if (u.u_procp -> p_pid == mon_accountant_pid
            || u.u_uid == MON_SUPERUSERUID)
        {
            if (!(mon_initflag))        /* not initialized    */
            {
                u.u_r.r_val1 = MON_NOT_INIT;
                mon_printf ("Not Initialized\n");
                break;
            }
            /*
             * close down all sensors
             */
            for (i = 0; i < MON_ENABLEVECTORSIZE; i++)
                mon_enablevector[i] = 0;
            mon_initflag = FALSE;
            mon_printf ("Sensors shut down by Accountant");
            u.u_r.r_val1 = 0;

            /*
             * reinitialize for next run
             */
            mon_write_ptr = mon_eventvector;
            mon_read_ptr = mon_eventvector;
            mon_eventvector_end = mon_eventvector + MON_EVENTVECSIZE;
            mon_eventvector_count = 0;
            mon_oflow_count = 0;
        }
        else
            u.u_r.r_val1 = MON_NOT_ACCTNT;
        break;

    default:
        u.u_r.r_val1 = MON_INV_CMD;
        mon_printf ("\n*******INVALID COMMAND****** CMD = %d\n",
            u_cmd_ptr->u_event.cmd.type);
    }  /* End switch */
    mon_printf("At end of call: mon_oflow_count = %d, mon_noreqflag = %d\n",
        mon_oflow_count,
        mon_noreqflag);
    mon_printf("at end of call: mon_enablevector = %x\n",
        mon_enablevector[0]);
    mon_semaphore--;
    return;
#endif MONITOR
}
```

```
/*
 *-------------------------------------------------------------
 *  Implements wraparound for the ring buffer.  The buffer
 *  has an extra tail equal to the length of the longest rec-
 *  ord, if data is written in here, it is copied onto the
 *  beginning of the file.  The reason for doing this is so
 *  that one doesn't have to check for the end of buffer on
 *  every byte of a record, only at end of write.
 *  It is the responsibility of the user to insure that the
 *  end of the tail is respected.
 *-------------------------------------------------------------
 */

u_char*
Wraparound(reg_ptr)
register u_char *reg_ptr;        /* position in ringbuffer */
{
#ifdef MONITOR
    register u_char *buf_end=mon_eventvector_end;  /* first pos of tail */
    register u_char *buf_start=mon_eventvector;    /* start of buffer   */

    if (reg_ptr > buf_end + MON_EVENTRECSIZE || reg_ptr < buf_start)
        panic("Wraparound: pointer out of range\n");

    while (  buf_end <= reg_ptr )
        *buf_start++ = *buf_end++;

    if (buf_start > buf_end + MON_EVENTRECSIZE || buf_start < mon_eventvector)
        panic("Wraparound: pointer out of range\n");

    return(buf_start -1);          /* next write position    */
#endif MONITOR
}
```

```
            mon_printf(("pos/16= %d, pos%%16= %d, &enablevec[pos/16]= %d\n
",
                    pos/16,
                    pos%16,
                    &mon_enablevector[pos/16]));
            u.u_r.r_val1 = 0;
        }
        else    /* for user preq->req.targetpid */
        {
            /*
             * Check for space in array
             */

            for (i = 0; (i < MON_REQLISTSIZE &&
                mon_requests[i].eventnumber != MON_REQOPENSLOT); i++)
                    ;
            if (i = MON_REQLISTSIZE)
            /* No room to store command */
            {
                mon_printf (("Command queue overflow!\n"));
                u.u_r.r_val1 = MON_REQ_OFLOW;
            }
            else    /* put in new commands for user processes */
            {
                mon_requests[i] = preq->req;
                u.u_r.r_val1 = 0;
            }
        }  /* else for user */
    }  /* if CALLERID */
    else  /* not the accountant or superuser */
    {
        u.u_r.r_val1 = MON_NOT_ACCTNT;
        mon_printf(("PUTREQ: not the accountant"));
        return;
    }
    break;

case MONOP_GETREQ:
/*
 * Read a request
 *  The request vector is searched for a request with
 *          a pid that matches the calling program.
 *  If the search is successful, copy the request back
 *          into the struct req in the command,
 *          clear the slot in the vector.
 *  else write an error record into the event vector.
 *  Return 0 if successful,
 *          MON_NOT_INIT if called before initialization,
 *          MON_REQ_NOT_FND if no request is found.
 */
    if (!(mon_initflag))            /* not initialized    */
    {
        u.u_r.r_val1 = MON_NOT_INIT;
        mon_printf (("Not Initialized\n"));
    }
    else
    {
        match = FALSE;
        i = 0;
        while (i < MON_REQLISTSIZE && !match)
        {
            match = ((int) mon_requests[i].targetpid == CALLERID);
            i++;
        }
        if (match)              /* Return command     */
        {
            mon_getreq      *greq = (mon_getreq *)u_cmd_ptr;

            i--;
            greq->req = mon_requests[i];
            mon_requests[i].eventnumber = MON_REQOPENSLOT;
            u.u_r.r_val1 = 0;
        }
        else                /* Can't find command     */
        {
            mon_printf (("Command not found in queue!\n"));
            if (mon_oflow_count == 0
               && mon_eventvector_count <
                   (MON_EVENTVECSIZE - 2*sizeof(mon_errrec)))
            {
                register mon_errrec *w = (mon_errrec *)mon_write_ptr;
                *w = mon_noreq;
                w->val = 0;
                mon_write_ptr += w->cmd.length*2;
                if (mon_write_ptr >= mon_eventvector_end)
                    mon_write_ptr = Wraparound(mon_write_ptr);
            }
            else mon_oflow_count++;
            u.u_r.r_val1 = MON_REQ_NOT_FND;
        }
    }
    break;

case MONOP_SHUTDOWN:
/*
 * Shut down monitoring
 * The accountant should close down all sensors and call GETEVENTS
 *          until it returns 0 before calling SHUTDOWNS,
 *          otherwise data will be left in the buffer and
 *          will be lost.
 * Closes down all kernel sensors, and prevents any other command
 *          other than init from being executed.
```