DISPLAY OF SEQUENT SYMMETRY S/81 PERFORMANCE USING X WINDOW SYSTEM FACILITIES

By

HAIBO DU

Bachelor of Arts

East China Normal University

Shanghai, China

1986

Submitted to the faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1993

OKLAHOMA STATE UNIVERSITY

DISPLAY OF SEQUENT SYMMETRY S/81 PERFORMANCE

USING X WINDOW SYSTEM FACILITIES

Thesis Approved:

M. Samadzadeh

Thesis Adviser

J. Chandler

Gary R. Brier

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to individuals who assisted me in this project and during my course work at Oklahoma State University. In particular, I wish to thank my major adviser, Dr. Mansur Samadzadeh, for his intelligent guidance, inspiration, and invaluable assistance. I am also thankful to Drs. John Chandler and Garry Bice for their advice and willingness to serve on my graduate committee. Their suggestions and support were very helpful throughout the study.

Many thanks are also given to my wife Hanzhen Yang and my son Zhouli Du for their moral support.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

A computer system performance report provides information important to its receiver, be it a system administrator or a user. It helps the administrator to keep the systems running efficiently by tuning it and it helps the user to make the best use of the system by user-level scheduling [Thobani 92]. The DYNIX/ptx operating system is compatible with AT&T UNIX System V3.2 and runs on the Symmetry S/81, Sequent's most powerful mainframe-class multiprocessor computer system [Sequent 90c]. DYNIX/ptx provides for the use of a command called **sar** that collects system-wide measurements including central processing unit (CPU) utilization, disk and tape input/output (I/O) activities, terminal device activities, buffer usage, system calls, system switching and swapping, file-access activities, queue activities, and message and semaphore activities [Sequent 91a]. A synopsis of the results of this command can be reported in a tabular format on the user's terminal screen. However, the use of the command is not as user friendly and the tabular reports not as readily understandable as one might desire.

Researches have shown that there can be a substantial difference between high and low degrees of user friendliness and interface in learning time, performance speed, error rates, and user satisfaction [Shneideman 87]. One solution to this problem is to use

graphical user interfaces (GUIs) and graphical display, which can help the user use the command more easily and assimilate the data more speedily. X Window System™ (often just called X) has a broad and powerful foundation for graphical windowing system, for it imposes very few restrictions on the development of high-level applications that support X, including window managers and toolkits. X provides mechanisms for implementing graphical user interfaces without imposing specific user-interface styles and hence accommodates the production of portable graphics as well [Smith 91]. Characteristics of X include portability and reusability [Barkakati 91], which are among the most important criteria of modern software development.

The main objective of this thesis was to implement user interfaces for graphical display at the user's terminal of system activity measurements obtained by applying the command **sar** under the DYNIX/ptx operating system on the Sequent machine using X Window System facilities.

Chapter II of this thesis report provides a literature review and context for Sequent S/81, Sequent System Activity Report Package, Graphical User Interface, X Window System, and OSF/Motif programming. Chapter III describes the implementation part of this thesis. Finally, Chapter IV contains a summary of the thesis, code characteristics of the **sarview** program, and some suggestions for future work.

CHAPTER II

LITERATURE REVIEW AND CONTEXT

2.1 Sequent S/81

2.1.1 Architecture

Sequent S/81 has a shared-memory, multi-processor architecture which comprises three elements: (1) a parallel architecture that employs multiple industry-standard microprocessors, (2) a variant of the UNIX operating system (that is becoming a worldwide standard for academic and commercial computing), and (3) standard interfaces like MULTIBUS [Sequent 90c].

The Symmetry S/81 system, that is available on the Computer Science Department at Oklahoma State University, has the following features in its present configuration [Sequent 90c]:

- 24 Intel 80386 microprocessors (expandable to 30), running at 20 MHz, each having a 128-kilobyte cache memory;
- Up to 384 megabytes of RAM with up to 43.3 gigabytes of hard disk space;
- Up to 256 directly-connected serial ports with up to four Ethernet ports;
- Up to 8 high-speed synchronous communication ports with up to 8 parallel printer ports; and

- Up to 4 half-inch reel-to-reel tape drives.

2.1.2 Operating System

The S/81 system can run one of the two operating systems, DYNIX V3.0 and DYNIX/ptx, both of which are complete UNIX system ports. DYNIX V3.0 is Sequent's dual-universe operating system, originally derived from UNIX 4.2 BSD. It supports both the Berkeley UNIX and UNIX System V command sets [Sequent 90c].

DYNIX/ptx is Sequent's implementation of the UNIX V operating system. It is compatible with AT&T System V3.2 (SVID Issue 3 compliant) and conforms to the IEEE Standard 1003-1988 Portable Operating System Interface for Computer Environments (POSIX), and also embodies the Federal Information Processing Standards Publication qualification and extensions to the POSIX Specification [Sequent 90c].

2.2 Sequent System Activity Report Package

The DYNIX/ptx operating system contains five commands related to the system activity report. They are **sar**, **sadc**, **sa1**, **sa2**, and **sag** [Sequent 91a]. The following subsections provide a brief description of these commands.

2.2.1 The sar Command

sar allows a user to generate system activity measurements in real time, to save the system activities in a file for later usage, and to display the data in tabular form at the terminal. The **sar** command facilitates the observation of system activity during [Perkin-Elmer 85]:

(a) A controlled stand-alone test of a large system,

(b) An uncontrolled run of a program to observe the operating environment, or

(c) Normal production operation.

The system activity information reported by **sar** is derived from a set of system counters located in the operating system kernel. This set includes counters for CPU utilization, buffer usage, disk and tape I/O activity, TTY device activity, switching and system-call activity, file-access, queue activity, inter-process communications, and paging (see Appendix B). These system counters are incremented as various system actions occur. They record various activities and provide the basis for the system activity reporting system. The data structure for most of these counters is defined in the sysinfo structure (see Appendix C).

As a system activity reporter, **sar** permits the user to specify a sampling interval as well as the number of intervals for observing system activity, and then to display the observed level of activity in tabular form. This system command can be used in two ways: **sar** [-ubdycwaqvmprA] [-o file] t [n] or **sar** [-ubdycwaqvmprA] [-s time] [-e time] [-i sec] [-f file] (see Appendix D). Using these options, **sar** generates (or displays) a set or subsets of data according to options specified. The twelve options, -ubdycwaqvmpr, each represent a subset of the sampled data (see Appendix D). The option -A represents the entire set of sampled data, that is, it is the same as -ubdycwaqvmpr.

Some specific cases involving some of the options of the **sar** command follow. If you want to see today's CPU activity so far, you may use **sar.** If you want to know CPU activity evolving over 20 minute intervals and save the results, you may use **sar** -o temp 60 20. If you want to review paging activities from that period on, you may use **sar**

-p -f temp [Sequent 91a].

2.2.2 The **sadc** Command

The command **sadc**, as well as **sa1** and **sa2,** can be used to routinely monitor and record system activity in a standard way for historical performance analysis. **sadc**, representing the system activity data collector, samples system counters for n times at every t seconds and writes the results to a binary file or to a standard output [Sequent 91a]. Its synopsis is:

/usr/lib/sa/**sadc** [t n] [ofile]

The sampling interval t should be at least 5 seconds (otherwise the activity of **sadc** itself may affect the sampling results). In case no t and n are given, **sadc** writes a dummy record in the file to indicate a system restart. An example would be the command entry in the etc/init.d/perf file [Sequent 91a]:

su sys -c "/usr/lib/sa/**sadc** /usr/adm/sa/sa/sa‘data +%d‘"

2.2.3 The **sa1** Command

DYNIX/ptx keeps its daily log file of system activity data in binary format in the directory /usr/adm/sa. The file is named sadd where dd stands for the digits representing the day of the current month. The command **sa1**, a shell script, is used to create the sadd file. Its synopsis is [Sequent 91a]:

/usr/lib/sa/**sa1** [t n]

A variant of **sadc**, **sa1** writes n times every t seconds when the arguments t and n are present, or once when the two arguments are omitted. A typical use would be the entries

in /usr/spool/cron/crontabs/sys which will write records at an interval of 20 minutes during working hours and hourly otherwise [Sequent 91a]:

```
0 * * * 0-6 /usr/lib/sa/sa1
20,40, 8-17 * * 1-5 /usr/lib/sa/sa1
```

2.2.4 The sa2 Command

This command is a shell procedure that invokes the **sar** command to generate daily report in file /usr/adm/sa/sardd. Its synopsis is [Sequent 91a]:

/usr/lib/sa/**sa2** [-ubdycwaqvmprA] [-s time] [-e time] [-i sec]

As a variant of **sar**, **sa2** accepts the starting and ending times as well as all report options of **sar**. The following entry in the file /usr/spool/cron/crontabs/sys will produce records hourly during a working day [Sequent 91a]:

```
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200
```

2.2.5 The sag Command

sag displays system activity data graphically. It depends on the data file generated by a previous run of **sar**. **sag** invokes **graphics** and **tplot** commands (see Appendix A) to have the graphical output displayed on the terminal types supported by **tplot**. On Sequent S/81, a Tektronix terminal type should be invoked to support this type of display. The following entry would display the current day's CPU utilization in xterm Tektronix mode [Sequent 91a]:

```
sag -T 4014
```

However, the execution of **sag** might not be as user friendly as one might expect or desire. For example, a user would have to key in all the required arguments. If any error

occurs, s/he would have to redo it. A user is also required to remember the data items to be displayed, because **sag** finds the desired data by string-matching the data column header [Sequent 91a].

2.3 Graphical User Interface

Much attention has been given to the subject of Graphical User Interfaces (GUI for short) in the past few years as they play an increasingly important role in modern application programs [Kobara 91] [Chandrashekar 93,91].

Graphical user interfaces were started at the Xerox Palo Alto Research Center (PARC) and were subsequently made popular by the Apple Computer Company in their Macintosh and Lisa systems [Barkakati 91]. Several such interfaces based upon the X Window System are available for UNIX, in addition to Microsoft Windows for MS-DOS personal computers and Presentation Manager for OS/2.

2.3.1 Definition

An application's appearance (look) and behavior (feel) is determined by its user interface. When a user interface makes use of graphical objects like windows and menus, it is said to be a graphical user interface [Barkakati 91]. It might also be called a point-and-click user interface, since users in general interact with a GUI by moving the mouse pointer on the screen and clicking. An example would be when a user clicks a mouse button with the pointer inside a box labelled "Done" to indicate consent to a choice.

2.3.2 Components

A GUI consists of four components: a window system, a window manager, a toolkit, and a style guide [Barkakati 91]. A brief description for each component follows.

Window System. The graphical window system manages output on the display screen and performs basic text and graphics drawing tasks.

Window Manager. The window manager offers the mechanism whereby users can indicate the window with which they want to interact when several windows appear on the screen. The window manager makes it possible for a user to move windows around and resize them in order to maximize the use of the limited size of the screen.

Toolkit. The toolkit is a library of routines with a well-defined programming interface, which is often referred to as the application programmers' interface (API). This API toolkit makes it possible for programmers to write applications that make use of the facilities of the window system and exhibit a consistent look and feel.

Style Guide. The fourth component, the style guide, specifies the appearance and behavior of the user interface of an application to help programmers follow a common set of guidelines so that the look and feel of applications built using a GUI may be consistent.

2.4 The X Window System

The window system is essential to any GUI. The X Window System™ (X for short) is among the most important software developments in the past few years [Smith 91] as it provides a broad and powerful graphical window system foundation for

implementing GUIs without imposing user-interface styles [Barkakati 91].

2.4.1 Definition

The X Window System is defined to be "a reasonably large software system that supports interactive computing among workstations" and "provides device-independent support for low-level, network-based windowing operations" [Smith 91].

2.4.2 A Brief History

The development of the X Window System started in 1984 at MIT by Project Athena [Barkakati 91]. The goal was to design a system to connect several different workstations in a network where graphical output could be sent transparently to any station [Berlage 91]. Under the auspices of external sponsors including DEC and IBM, this academic prototype was developed into a system suitable as a basis for commercial implementations. Several companies have joined to form the MIT X Consortium, which coordinates the future development of X and defines standards to improve compatibility among implementations [Berlage 91]. In September 1991, the Consortium released X11R5, its latest version by then [Scheifler 92].

2.4.3 Fundamental Components

X is different from older window systems in that it is not one homogeneous piece of software. Instead, it has three inter-related parts: a server, clients, and a communications channel [Mansfield 91].

Nye [Nye 90] defines the server as the software that manages the physical display

and its input devices such as keyboard and mouse. It is the server that actually creates windows, and draws images and text in the windows, in response to requests by client programs.

The clients are the application programs that utilize the system's window facilities. The reason to call the application programs "clients" is that they are, as a matter of fact, customers of the server in the sense that they require the server to perform specific actions on behalf of them.

The communications channel is used by clients to send requests to the server and by the server to send back status and other information to the client programs [Mansfield 91].

2.4.4 Client/Server Interaction

A client asks a server to execute a task (for instance, create a window with specified characteristics on a particular screen) by sending over the communication channel to the server what are called requests which are packaged in a simple block [Mansfield 91]. The block contains some code indicating what operation is to be performed, followed by a number of arguments providing more detail of what is required [Nye 90].

The server sends over the communications channel information back to the client about whether its requests were executed successfully, and informs the client on particular events which the client is interested in, which have occurred on the display [Mansfield 91]. Like requests from the client, these server responses are also packaged in simple blocks [Nye 90].

Events are fundamental to the client and server interaction. An event is a piece of information sent from the server to the client about a device performance or about a side effect of a previous client request [Nye 90]. All events are packed into a 32-byte structure to make it simple to queue and handle them. All keyboard input, and mouse button and motion inputs are handled using this event mechanism. Also, the client depends completely on events for information about certain occurrences on the system which it needs to know about [Mansfield 91].

Client/server communication falls into two broad categories that reflect the two basic modes of operating the X system [Mansfield 91] as explained below.

1. The client is running on the same hardware platform as the server. In this case, they can communicate via any method of inter-process communications available on the machine. When operating in this mode, X is effectively running like many conventional window systems.
2. The client is running on a different hardware platform from the server. Here, client and server must communication across a network using a mutually agreed protocol.

Several clients may be interacting with a single server, which would occur when several applications are displaying on a single screen, while a single client may also be communicating with several servers, which is the case when an announcement program is displaying the same message on several users' screens [Nye 90].

2.4.5 Multiple Software Layers

The X Window System™ is based on a distributed client/server model [Barkakati 91], and is implemented in multiple software layers. The hierarchy is presented in Figure

1.

The X protocol is at the lowest level. Considered as the machine language of X [Barkakati 91], the X protocol is designed to communicate all the information necessary to operate a window system over a single asynchronous bi-directional stream of 8-bit bytes [Nye 90]. Nye specifies four types of messages that can be transferred over the network in the X protocol: requests from the client to the server, replies, events, and errors from the server to the client. The protocol provides hardware independence for clients and servers (or stated differently, for applications and workstations) [Nye 90].

The use of asynchronous network protocol scheme instead of procedure or system calls provides a number of advantages [Nye 90]:

(1) Local as well as network connections can be handled in the same way using the protocol, making the network transparent to both a user and an application programmer.

(2) Implementation of the X protocol can be realized by using a wide variety of languages and operating systems.

(3) The X protocol can be used over any reliable byte stream via local inter-process communication or a network, some of which are standardized and available on most architectures.

(4) The X protocol causes little performance penalty on most applications, for performance is limited more by the time required to draw graphics than by the overhead incurred by the protocol.

Normally, clients implement the X protocol via a programming library that interfaces to a single underlying network protocol, typically TCP/IP or DECnet [Nye 90]. The sample implementation provided by MIT of the programming library is Xlib, the next

higher level to the X protocol.

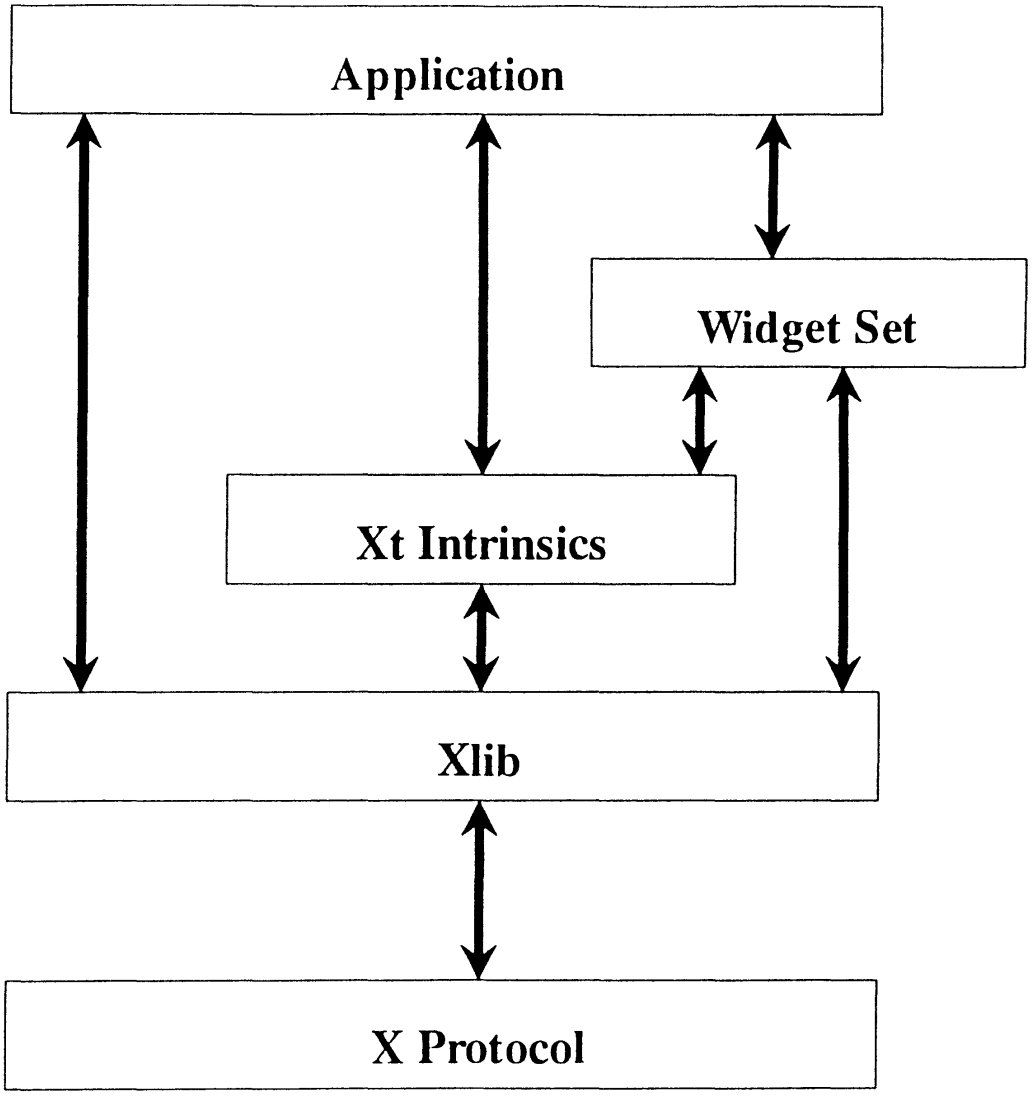


Figure 1. Hierarchy of software layers (Source: [Smith 91])

Xlib can be considered as the assembly language of X [Barkakati 91]. It is a C language-based client programming library that uses sockets on systems based on Berkeley UNIX, and provides code which simulates the sockets' interface on systems

based on AT&T's UNIX System V [Nye 90]. Xlib provides access to the X protocol through more than three hundred utility routines [Barkakati 91]. With Xlib, a programmer has a great amount of control over X applications. However, its capabilities are basic. It does not contain routines to create graphical objects such as menus and buttons. Although one can create an application with menus and buttons using a number of Xlib routines, it would be much more productive to have available a well-designed library of routines that can be used to create graphical user interfaces for any application [Barkakati 91]. Higher level interfaces, X Toolkit Intrinsics (also called Xt Intrinsics), have been developed for that purpose.

Xt Intrinsics can be considered the high-level languages of X [Barkakati 91], which takes an object-oriented approach to implementing basic building blocks known as widgets. A widget is defined as "a collection of data structures encapsulated with a public functional interface that implement a high-level GUI component, such as a menu" [Smith 91]. Widgets help to enhance reusability and data abstraction with respect to programming methodology. Xt Intrinsics provides functions applicable to all widgets and is a basis for other toolkits such as OSF/Motif, a well-known graphical user interface [Barkakati 91].

2.5 OSF/Motif Programming

OSF/Motif is a toolkit designed by members of the Open Software Foundation. The design goal was to provide the functionality necessary to implement graphical user interfaces that would perform identically on a wide variety of platforms, from high-end PCs to mini- and super-computers [Berlage 91].

2.5.1 Architecture

OSF/Motif has the toolkit of interface objects as its central feature with a number of other components that make the whole product work [Berlage 91], as shown in Figure 2.

As noted in Figure 2, the X Window System is the base layer of Motif, which provides many desirable features including responsibility for hiding machine-dependent differences as much as possible, allowance of multiple applications on different computers to be displayed in separate windows on the same screen, and coordination of the activities of the different applications that need not be aware of one another [Berlage 91]. However, X lets the applications totally control the contents of their respective windows, which makes it possible for environments with very different looks and feels to coexist. For example, you can run Motif applications and other X applications on the same screen, no matter how different they are [Berlage 91].

The Motif window manager aims at optimization for Motif applications [Berlage 91]. Besides handling moving and resizing, the manager manages the stacking order of overlapping windows, the iconization to reduce screen clutter. The Motif window manager also controls the input focus to determine which application window gets the keyboard input. The Motif window manager adheres to the Inter-Client Communications Conventions (ICCC), which makes it possible for X applications written with other toolkits to run equally well with this window manager [Berlage, 91].

The Motif toolkit, currently implemented as a C library, works with a variety of interface objects known as *widgets*. Typical widgets are push buttons, scroll bars or text entry fields. The functionality of the Motif toolkit can be divided into two: functions

applicable to all widgets, and functions needed to implement special widgets [Berlage 91].

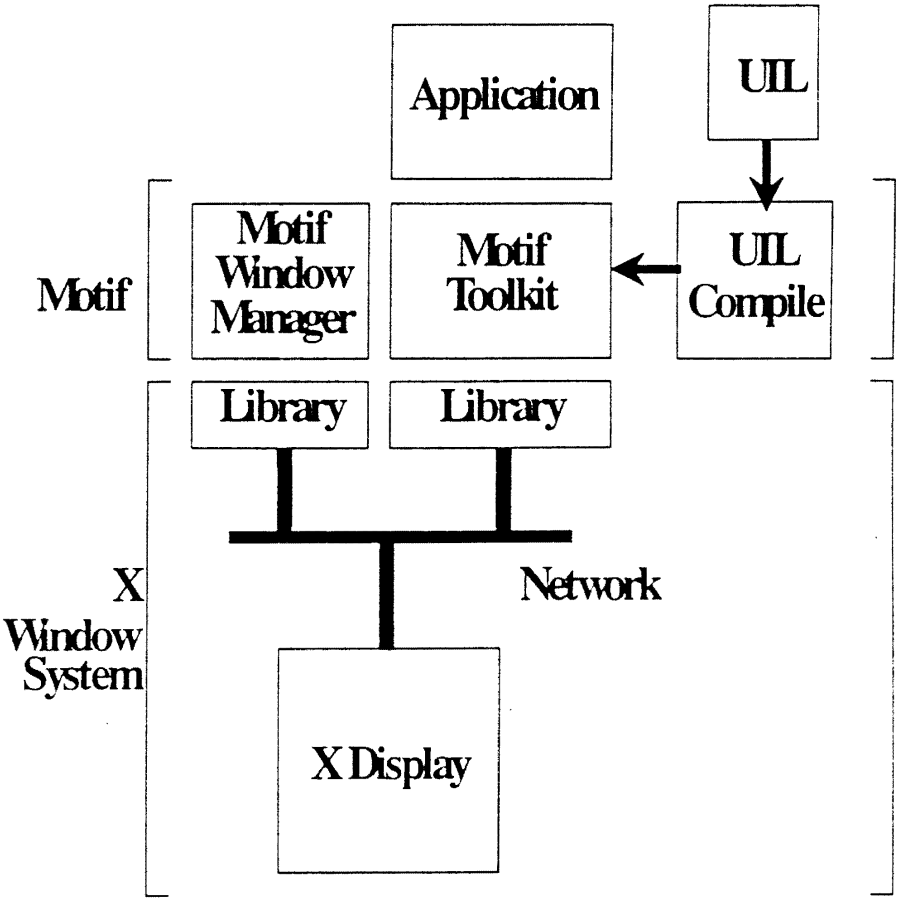


Figure 2. Architecture of OSF/Motif (Source: [Berlage 91])

UIL stands for User Interface Language. UIL allows an application designer to specify, outside of his/her program presentation, details such as the way menu entries are labelled, ordered, and arranged. These external specifications described in UIL can be modified without recompiling or relinking, since they are translated by the UIL compiler into an internal file format and are read in at run time when the program begins [Berlage 91].

2.5.2 Motif Widget Set

The Motif widget set groups widgets of the same type into classes that share the same program code. Instances of a widget class refer to the actual objects at run-time. Each of the instances has a separate storage for its attributes, but shares the common code. An example would be every button having a different string associated with it for display [Berlage 91].

The widget classes are further categorized into a class hierarchy, whose purpose is to share common features not only among instances of a class, but also among classes. Figure 3 shows the complete class hierarchy of Motif widget set. In this figure, class hierarchies go from left to right with a super-class to the left of its subclasses, whereas widget hierarchies are displayed in vertical direction with the parent on top of its children [Berlage 91].

The Motif widget set can be separated into three categories:

- Simple widgets that contain no children and are the basic building blocks, such as buttons and labels;
- Composite widgets that usually have other widgets as children, such as the menu bar, form, frame and pane; and
- Shell widgets that are always invisible because they always have the same size as their single child. Their only purpose is to perform as containers for other widgets. Shell widgets top the positions in an application's window hierarchy and are responsible for interacting with the window manager.

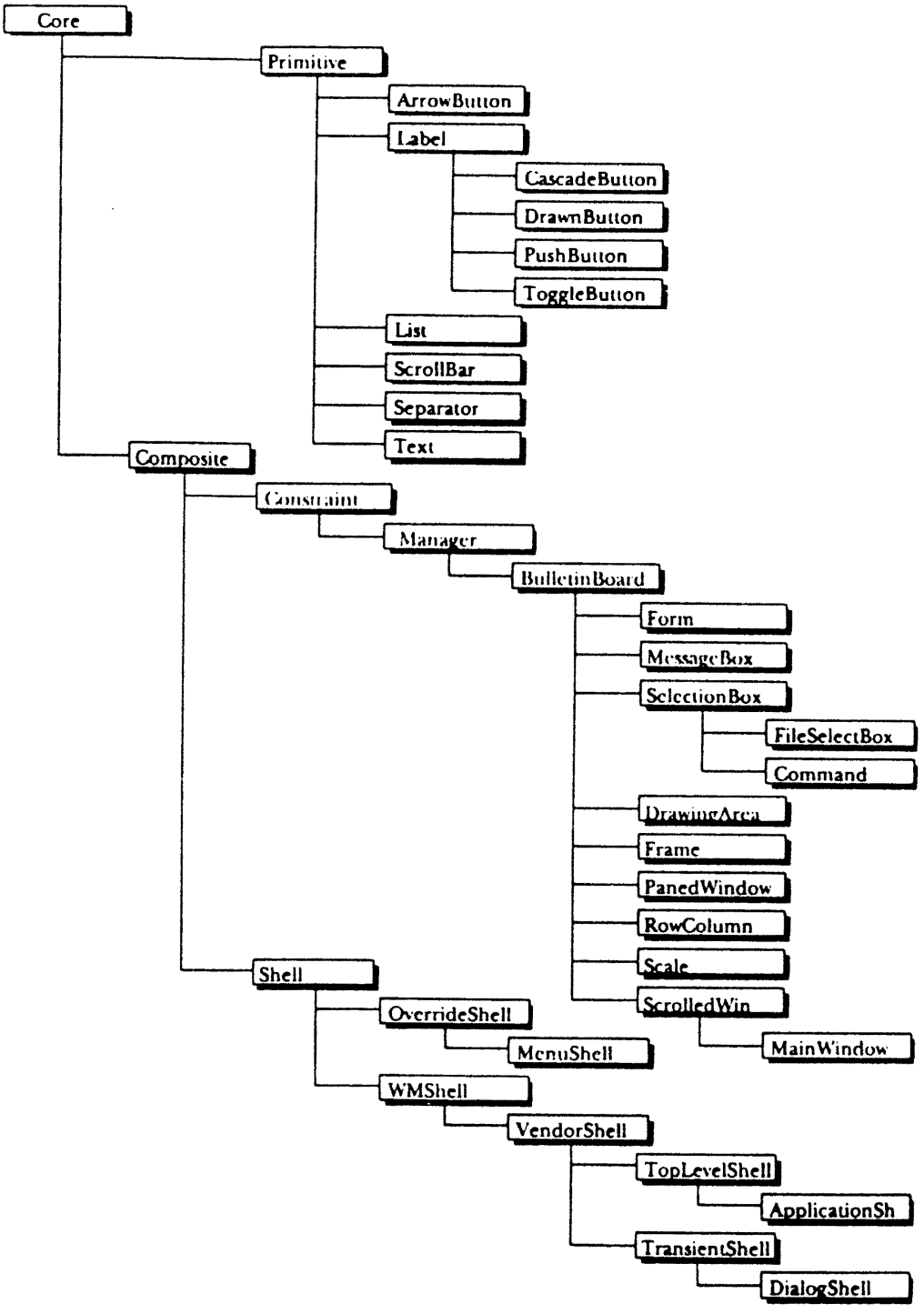


Figure 3. Class hierarchy of Motif widget set (Source: [Berlage 91])

2.5.3 Structure of a Motif Program

The skeleton structure of a Motif program's main procedure is shown in Figure 4. The beginning and end parts are standard more or less, while the middle part is application-reliant, since it contains the code to construct the widget hierarchy and to link it with the application code.

The main procedure can be divided into three phases: initialization, creation, and running.

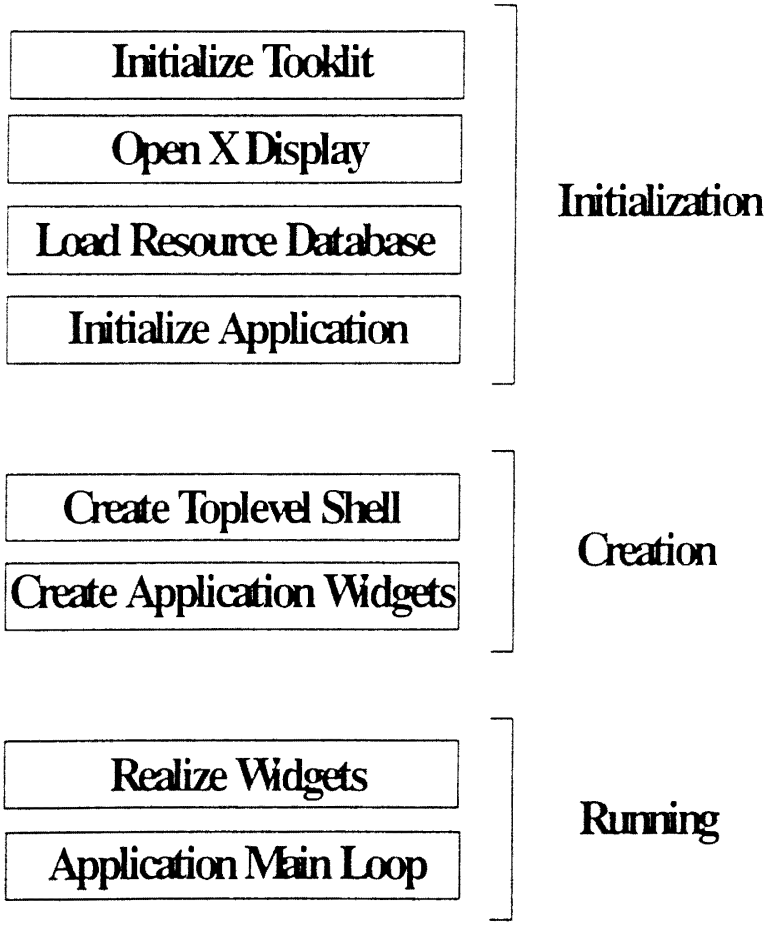


Figure 4. Structure of a Motif program (Source: [Berlage 91])

2.5.3.1 Initialization Phase. This phase has, in addition to application initialization, three steps: (1) initialize toolkit, (2) establish X display connection, and (3) load resource database [Berlage 91].

To initialize the X Toolkit, the following function is called:

```
XtToolkitInitialize();
```

This function call must be placed before all other calls to the X Toolkit or Motif widgets, but other initialization of the program can precede it.

Next called are the following functions to open the network connection to the desired X server.

```
context = XtCreateApplicationContext();
display = XtOpenDisplay(
    context, /* application context                */
    NULL,    /* use DISPLAY from environment       */
    NULL,    /* use last of argv[0] as name        */
    PROGRAM_CLASS,
    NULL,    /* no additional command line options */
       0,    /* ditto                              */
    &argc,
     argv    /* use and delete standard options    */
        );
```

The first parameter of XtOpenDisplay needs to be an application context, which is a vehicle to allow multiple logical applications to exist within a single physical program. Usually, one such application context is enough. The second parameter refers to the display specification. If it is NULL, the display specification is retrieved from the command line (option -display) or the DISPLAY environment variable, in that order.

Loading the resource database is realized through specification of the third and fourth parameters to XtOpenDisplay. If it is set to NULL, the application name is extracted from the command line (option -name) or from the last component of argv[0], that is, the name of the executable. Setting the application name to NULL offers further possibilities of parameterization to the user, who can set separate resource specifications

for the same program started in different ways. For instance, when initiated under the name color_demo, through renaming the executable or using the command line option

```
demo -name color_demo
```

the demo program is no longer influenced by the resource specifications associated with "demo". In this case, a complete set of resource definitions is kept in parallel for both the "demo" and the "color_demo" invocations.

A Motif program's class name should be constant and defined in the program. This example specifies the name using a preprocessor definition

```
#define PROGRAM_CLASS "Demo1"
```

As a convention, a class name should begin with an upper-case letter.

The last two parameters of XtOpenDisplay can be used to pass a number of standard command line options recognized by XtOpenDisplay, while the fifth and sixth parameters can be used to define the programmer's own options.

One necessity after calling XtOpenDisplay is to check the returned value. XtOpenDisplay returns a pointer to the toolkit's internal display structure, declared at the start of the main procedure as

```
Display    *display;
```

The NULL pointer indicates failure in establishing the display connection, which may result from various situations [Berlage 91] as sketched below.

- The user has failed to specify a display (no DISPLAY environment variable and no command line argument).
- The display specification is incorrect.
- No X server is running on the machine concerned.
- The host on which the program is running has not been granted access rights to the X

server.

When a connection failure occurs, the program would typically report an error and terminate.

2.5.3.2 Creation Phase. The creation phase of the main procedure contains two steps: creating a top-level shell and creating the application widgets.

Each application has to create a top-level shell widget and has that widget serve as a root for the widget tree and all resource specifications. The creation procedure follows:

```
XtSetArg(args[0],XmNallowShellResize,True); /* optional  */
toplevel = XtAppCreateShell(
         NULL,                /* use same program name    */
         PROGRAM_CLASS,       /* use same class name      */
         applicationShellWidgetClass,
         display,
         args,1               /* argument list            */
         );
```

The first two parameters to XtAppCreateShell refer to the application name and class, which are the same as to XtOpenDisplay. The third parameter defines the widget class. The symbolic name applicationShellWidgetClass is defined in "Shell.h" as a pointer to the class record of the application shell class. Although other shell classes may be used for top-level shell, this is not recommended, according to Berlage [Berlage 91].

The display variable is passed for the purpose of determining which root window is used for this application. The last two parameters are used to specify an argument list to provide initial values for some resource fields of the new widget.

The step after the top-level shell has been created is Create Application Widgets. It creates other widgets as sub-trees of this shell. For programs with only a few widgets, the simplest way is to create all widgets at start-up, even including the dialogue boxes

that are popped up when required. In large programs with many widgets, parts of the widget tree can be created on demand to avoid an otherwise significant start-up delay.

There are three possible procedures that can be used to create a widget, as listed below.

(1) Creating a widget unmanaged using XtCreateWidget.

(2) Creating a widget managed using XtCreateManagedWidget.

(3) Creating a widget using one of the Motif convenience functions.

The synopsis for the first procedure follows [Rost 90].

```
Widget  XtCreateWidget(name,class,parent,args,num_args)
     String       name;  /* name for the created widget    */
     WidgetClass class;  /* widget class pointer for the   */
                         /* created object                 */
     Widget      parent; /* parent of the created widget   */
     ArgList     args;   /* list of arguments that         */
                         /* override resource defaults     */
     Cardinal  num_args; /* number of arguments in         */
                         /* argument list                  */
```

This XtCreateWidget routine creates an instance of a widget of the particular class and returns the ID of the widget created. The first parameter of this routine specifies the widget name, which will be used to get resources from the resource database for the widget. The widget name can be freely chosen and, as a convention, should start with a lower-case letter.

The next parameter of this procedure defines the class identifier, which must be equal to or a subclass of the Core widget class. This class identifier is activated as a pointer to the class record of the widget class.

The third parameter specifies the parent widget, which every widget must have. From the specification of the parent during a widget creation, we can see that the widget hierarchy is constructed strictly top-down.

The last two parameters of this function specify program-determined resource field values for the widget to be created and the number of arguments, respectively. The resources specified in args will be used to override the default ones.

An example call to create a simple toggle button widget called "button" is given below.

```
button = XtCreateWidget(
    "button",
    xmToggleButtonWidgetClass,
    parent,
    args,0
    );
```

The parameters for the XtCreateManagedWidget are the same as those for the XtCreateWidget. The Motif convenience functions have two of their parameters in exactly the reverse order as compared to those for XtCreateWidget [Berlage 91].

2.5.3.3 Running Phase. This phase includes two stages, one to realize the top-level shell and the other to execute the main event loop. The realization of the top-level shell widget is done by a single call

```
XtRealizeWidget(toplevel);
```

This routine realizes all the child widgets of the top-level shell in a bottom-up fashion. Thus the top-level shell is the last to be realized. Those widgets with the mapped-when-managed flag will be mapped and thus visible on the screen.

After the widgets have been realized, the program enters the main loop to process the events generated by the X server or by user actions. Entry into the main loop is invoked by a call to the following routine

```
XtAppMainLoop(context);
```

This XtAppMainLoop function handles application-specific actions through callback procedures registered with a widget. The synopsis to register a callback procedure follows

[Rost 90].

```
void XtAddCallback(widget,callback_name,callback_proc,client_data)
     Widget; /* widget whose callback list is to be modified     */
     String callback_name;    /* name of callback list to which   */
                               /* to add callback                 */
     XtCallbackProc callback;/* callback procedure to be          */
                              /* added to callback list          */
     XtPointer client_data; /* argument to be passed to callback  */
                            /* procedure when it is called       */
```

This function adds the specified procedure to one of the widget's callback lists, which is specified by callback_name. The registered callbacks are invoked at a time determined by the widget's semantics [Rost 90].

The following example registers a callback procedure called Quit for XmNactivateCallback:

```
XtAddCallback(button,XmNactivateCallback,Quit,NULL);
```

XmNactivateCallback is the reason for Quit to be invoked when the button widget is pressed. You can register more than one callback procedure for a single callback reason by adding it to the list. You can also install the same procedure with different callbacks of different widgets. The above two approaches to register are made possible by the ability of the callback procedure to adjust its behavior according to its parameters.

All callback procedures, which are handled by common routines, have identical parameters [Rost 90], as shown below.

```
void Quit(widget,client_data,call_data)
     Widget;
     caddr_t client_data;
     XmAnyCallbackStruct *call_data;
```

This Quit callback routine has three parameters. The first parameter always indicates the widget for which the callback was registered. This parameter provides a programmer with the capability to examine the resource fields of the calling widget and alter them if necessary. The second parameter of the above routine is a pointer, which simply passes

the value registered with XtAddCallback. By referring to the client_data, you can specify additional private parameter values. The third parameter is also a pointer, which points to a structure specific to the callback reason. For example, each activation callback of each toggle button passes a reference to an XmAnyCallbackStruct.

To exit a Motif program, two common methods are applied. One is to define a callback procedure that executes an exit call in the program. Termination of a program in this way is usually actualized through pressing a button or menu entry labeled "Quit" or "Exit" in one of the program's windows. The other is to use the Motif window manager's close function, which is invoked by double-clicking on the window menu button positioned in the upper left corner of the window manager decoration.

CHAPTER III

IMPLEMENTATION

This chapter describes the design and implementation of the performance display program. The complete code and necessary support files are given in Appendix G.

3.1 Overview

3.1.1 Purpose

The purpose of the performance display program (called **sarview**) is to serve as a graphical displayer for Sequent S/81 system performance. It uses the information generated by **sar**, but presents it graphically in a more user friendly way.

3.1.2 Program Structure

This performance display program is divided into three parts. One part deals with loading and maintaining the information to be presented graphically. This part is reasonably independent from the user interface design and thus can be treated separately. The second part is the code for the actual user interface. The third part is the code responsible for the graphical display. Such a separation helps in ensuring greater modularity. The three parts are contained in three different files, getdata.c for the data

access layer, menu.c for the interface, and graph.c for the display (see Figure 5).

For global variables and type declarations, all three files share an include file called sarview.h. The include file motif.h contains header files for X and Motif, which is shared by menu.c and graph.c. The menu.c component also has its own include file called menu.h and a supporting file called des.text. The graph.c component has its own include file called graph.h and a supporting file called help.text. des.text provides the contents for the description window and help.text provides the contents for the help window.

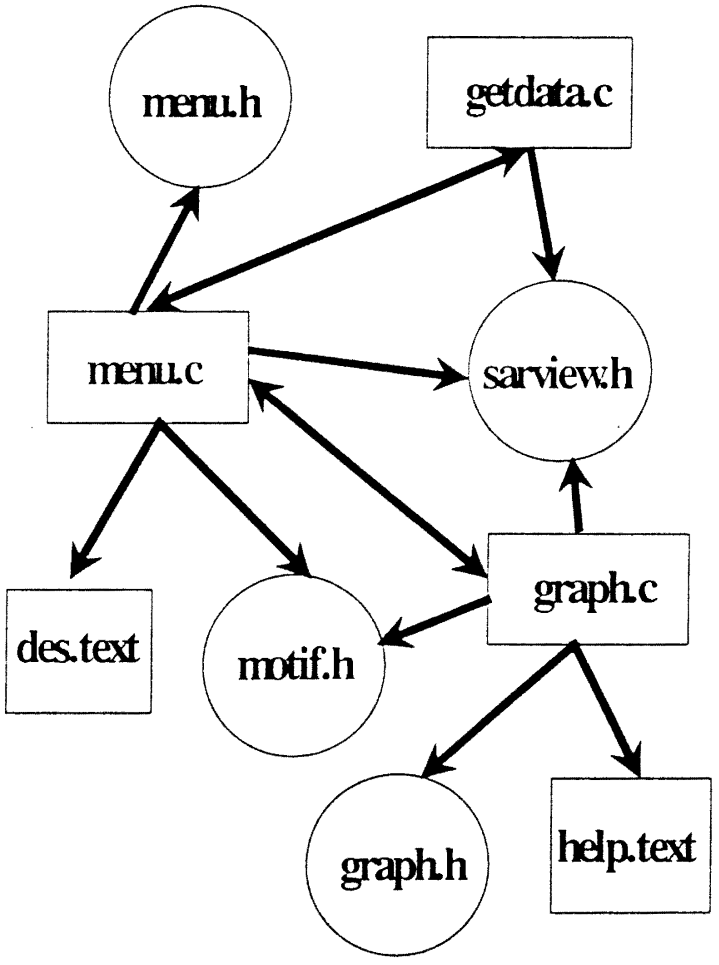


Figure 5. Interdependence among the main modules of the **sarview** program

3.2 Data Access Layer

The procedure get_sar_data in sardata.c is responsible for reading in data when information about a particular file is requested. It first calls get_line_number, which runs an **sar** command (for a description of the **sar** command and its options see Section 2.2.1) with the data file specified by the user to get the data size for the subsequent memory allocation. It then invokes a shell script as shown below.

```
/bin/csh -fc '(sar [-ubdycwaquvmprA] [-s time] [-e time] [-i sec] \
        [-f file] > /tmp/CC.stdout) >& /tmp/CC.stderr'
```

which executes the command **sar**, redirecting the standard output into a file called CC.stdout and the standard error into CC.stderr if any, both under the directory /tmp. If there is no error message (size of CC.stderr is zero), data in CC.stdout is read in and filled into a memory resident structure called Sa. If an error occurs during data access, the program will terminate with an error message posted.

The main performance data structure is described in Figure 6. Sa contains pointers to twelve structures for the twelve options. This scheme allows a dynamic buffer allocation for data to be read during the program execution.

Information on an **sar** option is read into a buffer (corresponding to that option) by invoking a corresponding routine. For example, the cpu utilization option would call a function named get_cpu_data, which reads the corresponding data into a structure referred to as CPU_Utilization. This data structure is defined as follows.

```
typedef struct {
  char time[LENGTH];
  char user[8];
  char sys[8];
  char wio[8];
  char idle[8];
  } CPU_Utilization;
```

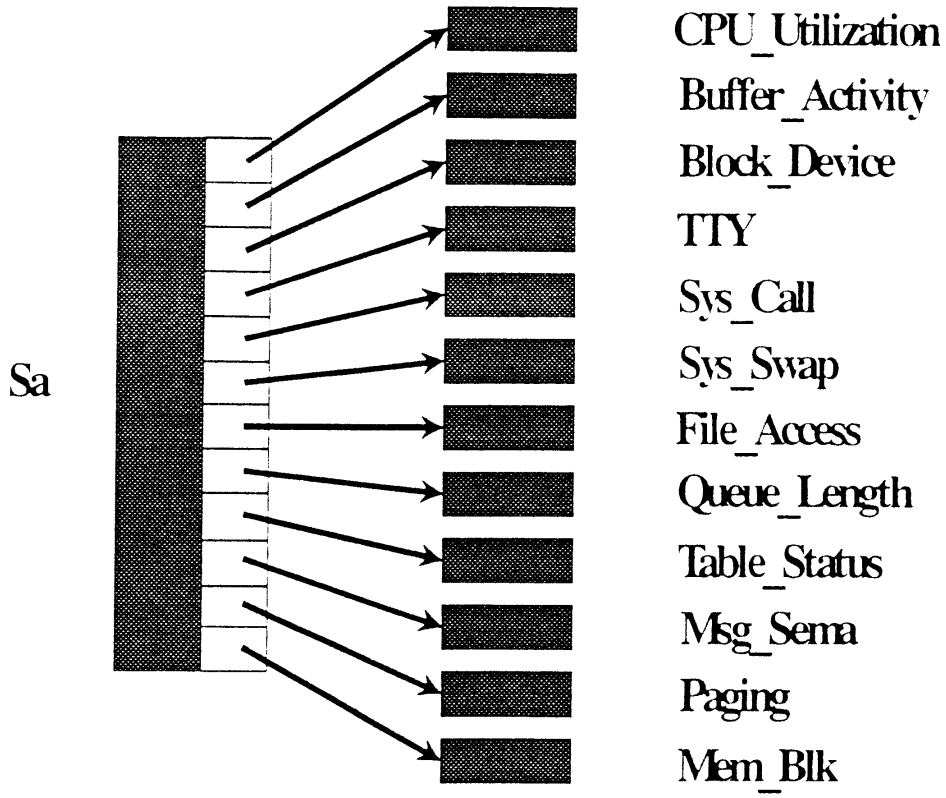


Figure 6. Internal data structure for **sar** information

The data structures for the other options are similar to this data structure except for some of their fields which are defined differently for each data structure (see Appendix G).

The above-mentioned get_cpu_data function returns zero if data access is successful, and nonzero otherwise. The other details about the data input (essentially consisting of a number of data transfers) are rather trivial compared with those of the user interface, and are thus omitted.

3.3 The User Interface

The code in the menu.c file handles the performance display program's graphical user interface layer.

3.3.1 The Main Procedure

The initialization and running phases of the main procedure are similar to those described in Chapter II (Section 2.5.3). Application initialization includes two functions

```
app_init();
busy_cursor = create_busy_cursor(toplevel);
```

The former initializes all the necessary parameters and the latter creates a cursor indicating "busy" for later use.

Creation of application widgets is achieved by calls to the following two routines.

```
menu = create_menu(form);
create_description_w(form,menu);
```

create_menu creates and manipulates widgets for the menu interface and functionality, returning the widget ID of the menu; create_description_w creates and manages a label widget and a scrolled text widget for the program description window.

3.3.2 The Widget Hierarchy

The widget hierarchy of the performance display program (**sarview**) is depicted in Figure 7. The widgets are created in a top-down fashion. Each widget represents an instance of its widget class. Some widget classes are Xt toolkit based and others are Motif specified.

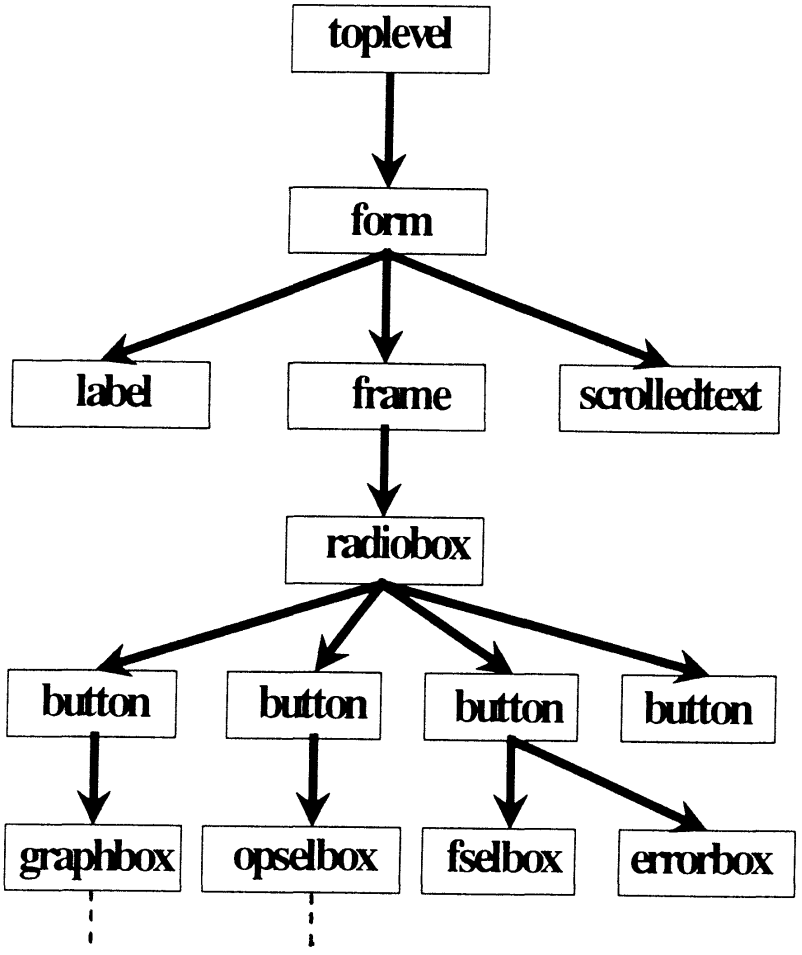


Figure 7. Widget hierarchy of the **sarview** program

3.3.3 Interface Objects

The visible components of the **sarview** program are the main menu window, three dialogue boxes, and up to twelve displaying windows, plus two message boxes and one prompt box that are popped up upon demand. Each component consists of one or more widgets on the hierarchy tree, as illustrated in Figure 7.

3.3.3.1 The Main Menu Window. The main menu window corresponds to the following list of widgets on the hierarchy tree in Figure 7: form, label, frame, scrolledtext, radiobox, and buttons. As seen in Figure 7, the form widget is created as the single child of the toplevel shell and as the container for other widgets in the window. Its width and height are set through calls to the following two routines.

```
XtSetArg(args[n],XmNwidth,700); n++;
XtSetArg(args[n],XmNheight,222);n++;
```

The three child widgets of the form widget are arranged such that the frame widget appears on the left of the main window as the menu pane and the scrolledtext widget on the right as the description window with the label widget on its top. The radiobox widget is created as the child of the frame widget to facilitate the layout and look of the four toggle button widgets representing the menu entries (i.e., File, Options, Graphics, and Quit). Particular callbacks are registered with each of the four button widgets so that pressing the File, Options, or Graphics buttons will pop up a corresponding dialogue box. Pressing the Quit button will terminate the **sarview** program.

Figure 8 shows the main menu window. The title bar indicates the function of the program. The description window provides some information about the program and some helpful hints. The scroll bar in the window can be slid vertically so that text longer than the viewable area can be scrolled into view. This saves space in a cluttered screen and allows large amounts of text to be viewed in a small space. If the window is made longer than the underlying data, the scroll bar will disappear. This is because the resource for the description window, XmNscrollBarDisplayPolicy, is set to XmAs_NEEDED. In contrast, if the window is forced to be narrower than the underlying data, a horizontal bar will appear to make it possible for the text to be scrolled horizontally. The four toggle

buttons are arranged vertically in a box, which looks like a radio box.

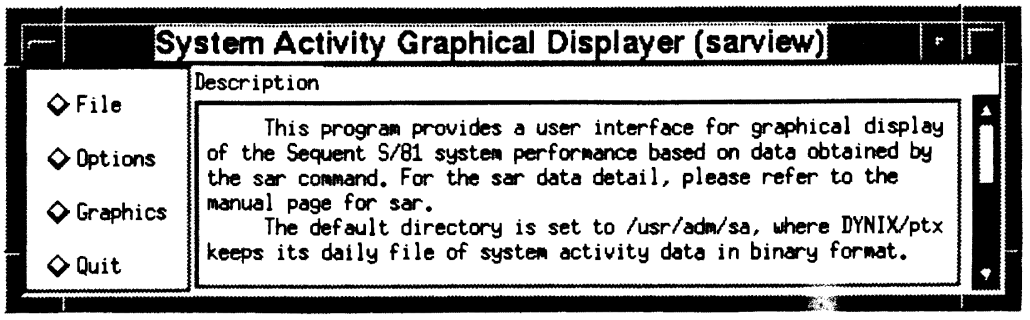


Figure 8. Main menu window

3.3.3.2 The File Selection Window. When clicked by a mouse, the File button is toggled to give a look of being depressed as shown in Figure 9, and a window titled File Selection pops up, as shown in Figure 10. This window is created by using a Motif convenience function called XmCreateFileSelectionDialog. The dialogue style of this window is set to XmDIALOG_APPLICATION_MODAL, so that the user must respond to the dialogue box before any further interaction can be done in this program. Such a specification is reasonable because selection of a file is usually the first thing to do.

The File Selection dialogue window prompts the user to select a file. The default directory is set to /usr/adm/sa, where DYNIX/ptx keeps its daily file of system activity data. A file will be highlighted when it is selected (the file /usr/adm/sa/sa18 is highlighted in Figure 10). The user can change to another directory through the File Filter entry or can directly choose a file through the Selection entry. When the commitment button OK is clicked, indicating confirmation to the selection of a file, the window is popped down and the File button in the main window toggles back. However, if the chosen file is of

any type other than a data type, the window will not be popped down. Instead, an error message box will pop up to echo the error, and the user must respond to it before any further interface can take place (see Figure 11).

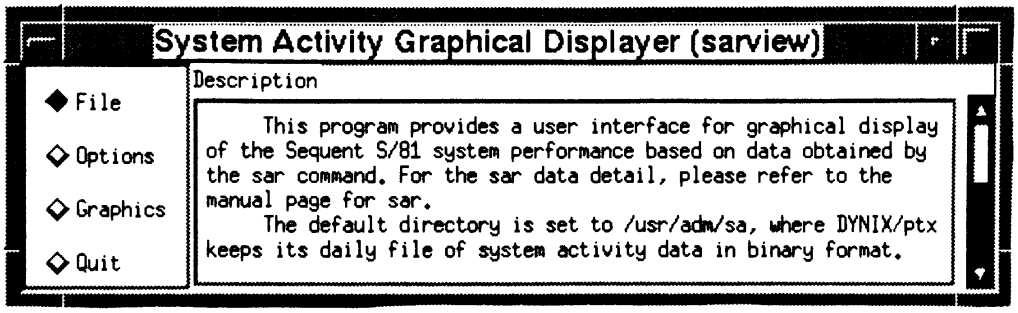


Figure 9. Pressed File button

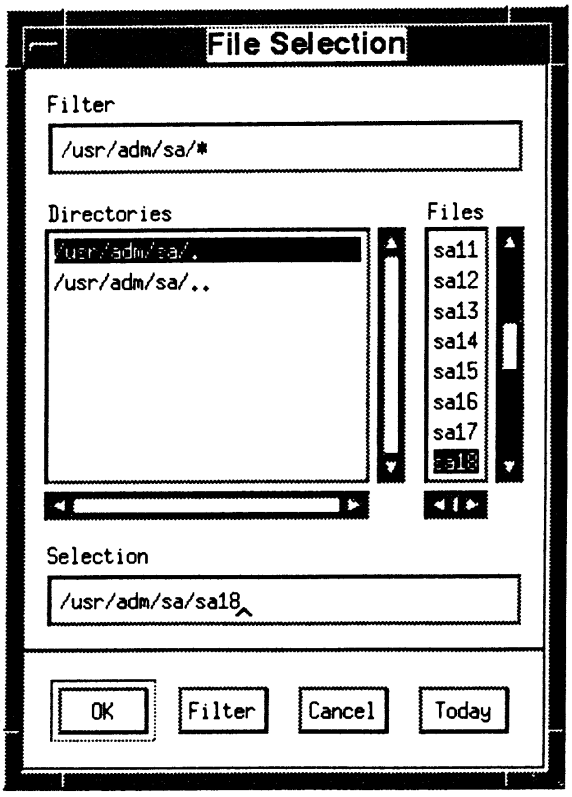


Figure 10. File selection window

Pressing the button Today selects the file for today's system performance, which is the default file of the command **sar**.

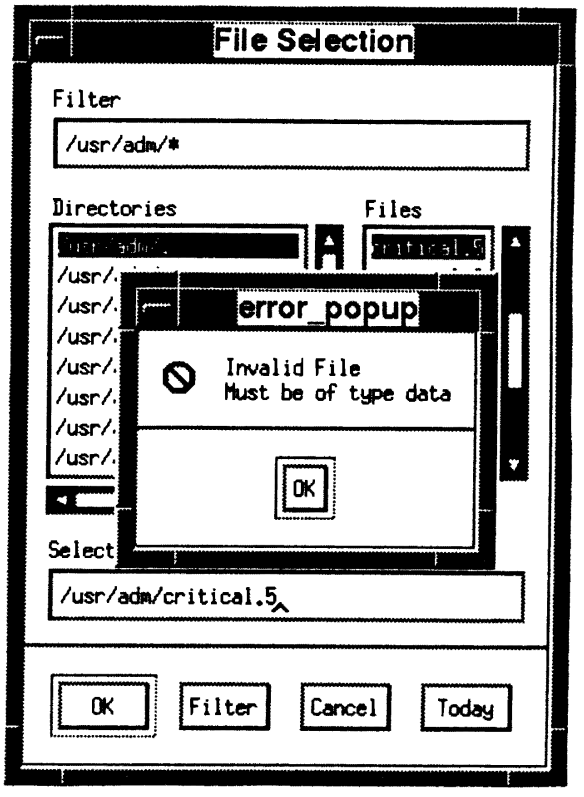


Figure 11. Error message box

3.3.3.3 The Options Window. After selection of a file, the button named Options in the main menu window is pressed, resulting in the toggling of the button (see Figure 12) and the popping-up of a dialogue box called Performance Options as shown in Figure 13. The widget hierarchy tree of this Performance Options box is outlined in Figure 14.

From Figure 14, we see that bulletinboarddialog is a shell widget, which is created by using a Motif convenience function known as XmCreateBulletinBoardDialog. The dialogue style of this shell is set to XmDIALOG_APPLICATION_MODAL, similar to

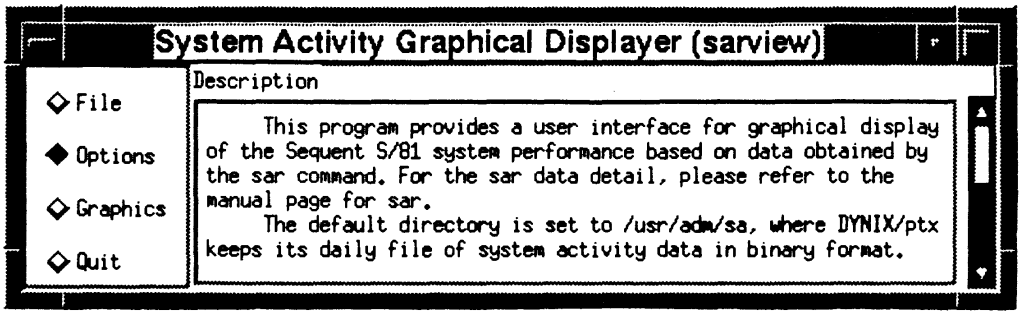


Figure 12. Pressed Options button

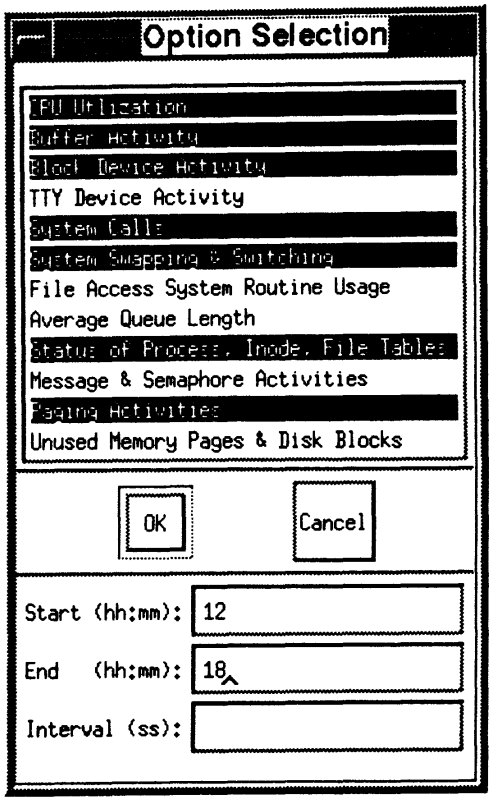


Figure 13. Option Selection window

the one in the File Selection Box. After the bulletin board shell is created, the pane widget is created as the child of the dialogue shell and as the container for the other widgets. The pane widget has three widgets as its children: a scrolledlist and two forms. The scrolledlist widget is used to make the multi-selection box, which displays the 12 options to be selected. One form widget is used to arrange its two children: push-button widgets OK and Cancel. The other form widget serves as a container for and organizer of its children: three form widgets, which make up the selection pane for start time, the end time, and the amount of the interval. Through this Options Selection window, the user can choose from among the twelve options concerning the system performance.

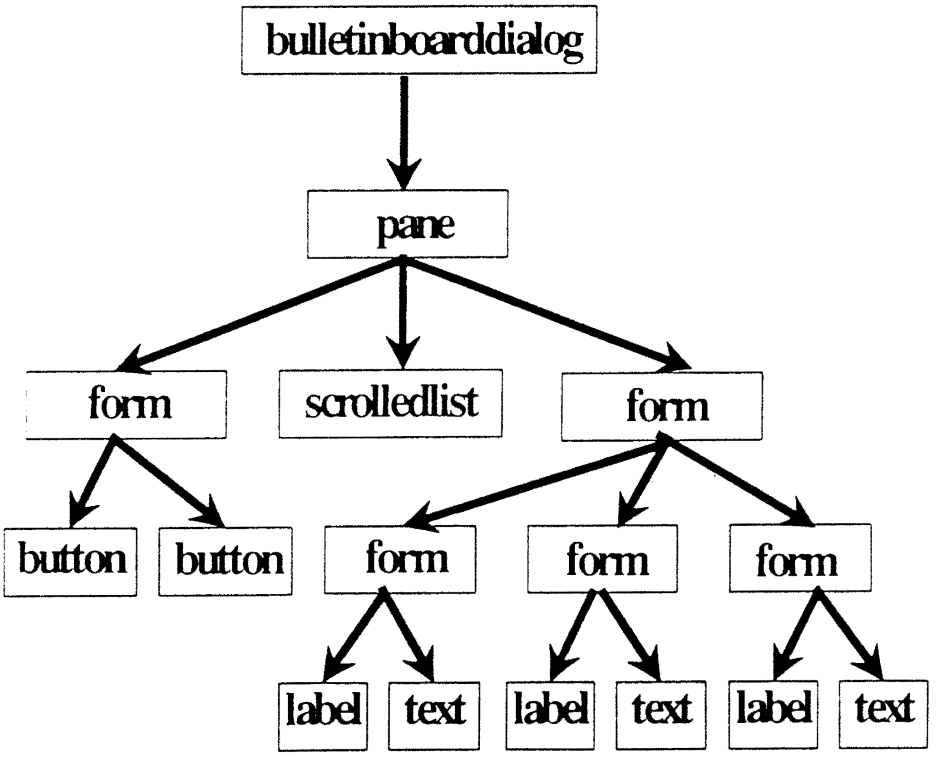


Figure 14. Widget hierarchy of the Option Selection window

The chosen item in the Options Selection window is highlighted analogous to the one in the File Selection window. The user can also specify the start time and the end time, as well as the interval amount, of the performance report, through the specification pane at the bottom of this Options Selection window.

When the selection is finished, OK is pressed, resulting in the pop-down of the window and the toggle-back of the Options button in the main window.

3.3.3.4 The Graphics Selection Window. Figures 15 and 16 show the windows after the Graphics button is clicked. The widget hierarchy of the Graphics Selection window is diagrammed in Figure 17. The top two levels are the same as those in the Options Selection window. The pane widget has three children. One is the message box, which provides on-line help hints. One is the radiobox, which holds the twelve toggle buttons in a vertical direction. The third is the toggle button for quit.

From Figure 16, we can see that five options look grayed, which are the items not selected in the Options Selection window, such as TTY Device Activity and Average Queue Length. The faded options will be insensitive to the user's input.

3.3.3.5 The Graphics Display Window. Pressing one of the 12 buttons in the Graphics Selection toggles the button and pops up a window for graphical display of the corresponding item.

There are two kinds of drawings in the display window: direct and indirect. By direct, it is meant that the graphs show up when the display window is popped up. When it is indirect, the display window that has been popped up remains blank until the user presses one of the drawing buttons. Two of the options take the indirect approach, namely CPU Utilization and Block Device Activity. The rest of the options take the direct

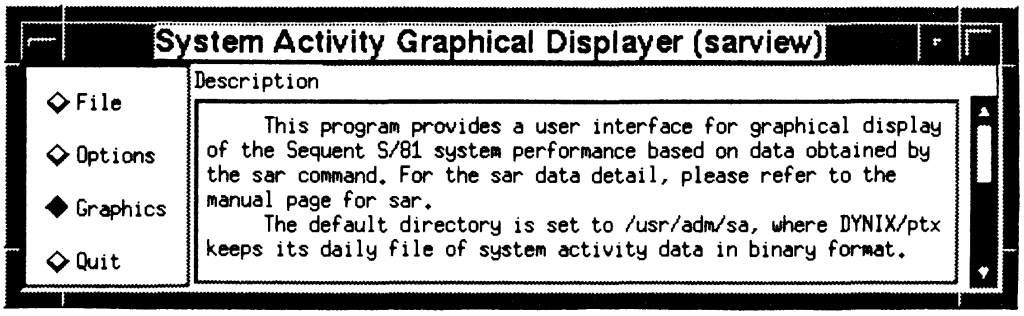


Figure 15. Pressed Graphics button

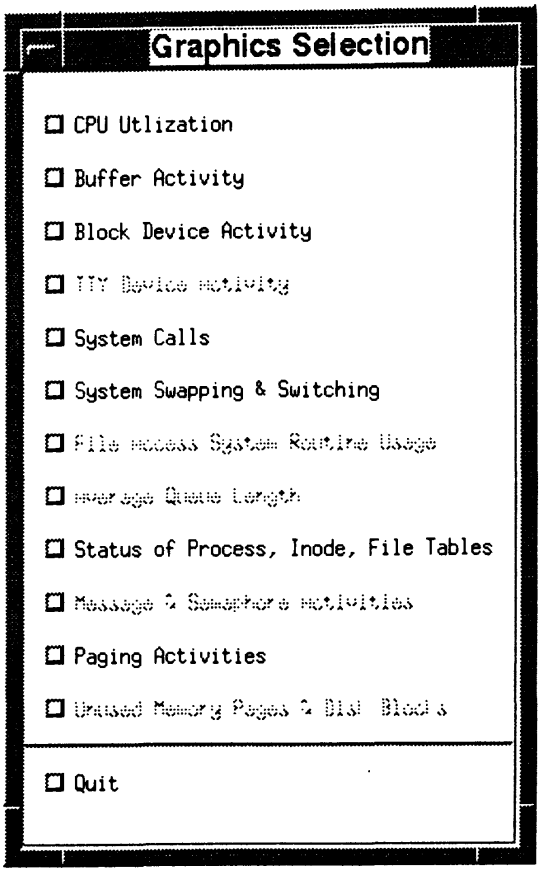


Figure 16. Graphics Selection window

approach. Since drawing takes time, at the time the user asks for drawing, the mouse pointer is changed to a busy cursor, as a quick response to the user's action, until the drawing is displayed.

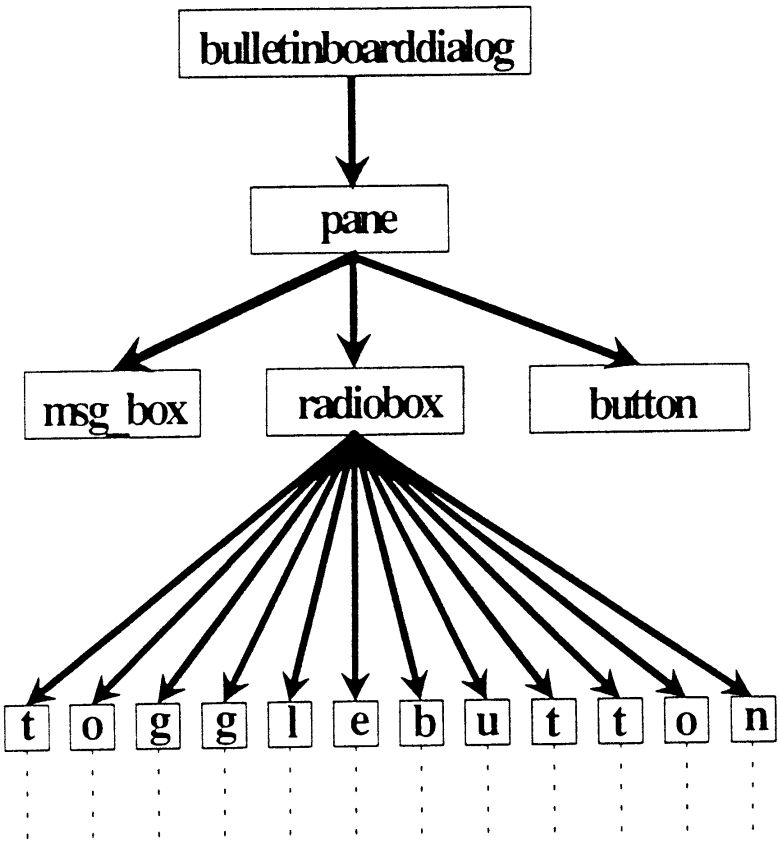


Figure 17. Widget hierarchy of the Graphics Selection window

A typical widget hierarchy of such a window is depicted in Figure 18. At the top of the hierarchy is the formdialog widget, which is created by using a Motif convenience function called XmCreateFormDialog, as the child of its corresponding toggle button in the Graphics Selection window and as the container for the other widgets in the display window as well.

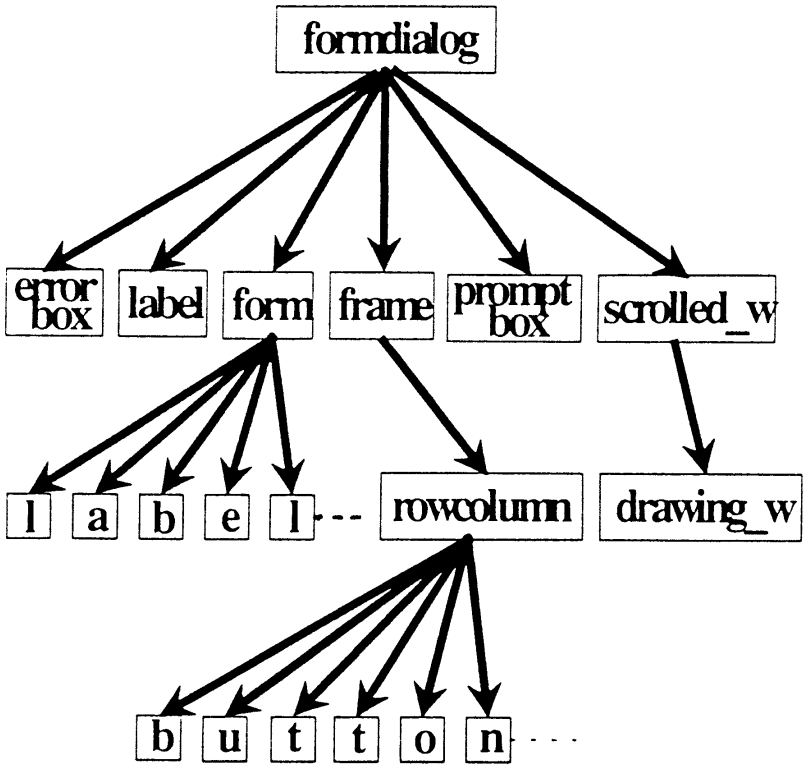


Figure 18. Widget hierarchy of the drawing window

The formdialog widget has six children, namely error box, label, form, frame, prompt box, and scrolled_w (scrolled window). The error box is popped up when no file is specified for image saving in the prompt box. The label widget displays information about the host name and the date of the performance data collection. The form widget is used to manage a number of label widgets, which form the legend area at the bottom of the window.

The frame serves as the menu pane and has as its child the rowcolumn widget, which manages the push buttons. Three or more push buttons are used based on the individual display windows. The common buttons are Help, Save, and Quit. The Help

button provides an on-line explanation about the data items displayed in the graphic charts. Pressing the Save button pops a prompt box that asks the user for the name of a file, into which the corresponding drawing window is to be dumped.

The scrolled window has as its child a drawing widget called drawing_w, which is bigger than its parent. The contents of the drawing widget beyond the scrolled window can be viewed through sliding the scroll bars vertically or horizontally.

When finished with the display, the user can press the quit button to pop the display window down.

3.3.4 Graph Drawing

The file graph.c takes care of the graphical display part of this program (see Figure 5 and Appendix G). The data for the bar and pie chart in the display window is extracted from the data structure Sa described in Section 3.2. The drawing into a window is accomplished by utilizing the drawing functions of Xlib such as XDrawArc, XDrawRectangle, XDrawImageString, and XFillArc.

Several functions are dedicated to the bar chart drawing. The draw_rectangle procedure is responsible for drawing the chart box, the draw_scale procedure for the chart scale, and the draw_time procedure for the time drawing of the chart's y axis.

In order to facilitate the quick and smooth exposure of a graph window during scrolling, pixmap is used. Data related to a particular item is first drawn onto a pixmap. Then the pixmap is copied onto the specified window. Every time an exposure request is invoked, the corresponding pixmap is used and copied onto its corresponding window.

3.3.5 Example Display Windows

Three display windows are chosen to represent the features of all the display windows since the look and behavior of the rest of the display windows are similar to those of the chosen ones. The three are the windows for CPU Utilization, Buffer Activity, and Block Device Activity.

Pressing the CPU Utilization button in the Graphics Selection window (see Figure 19) pops up a window for presenting CPU utilization graphically as shown in Figure 20.

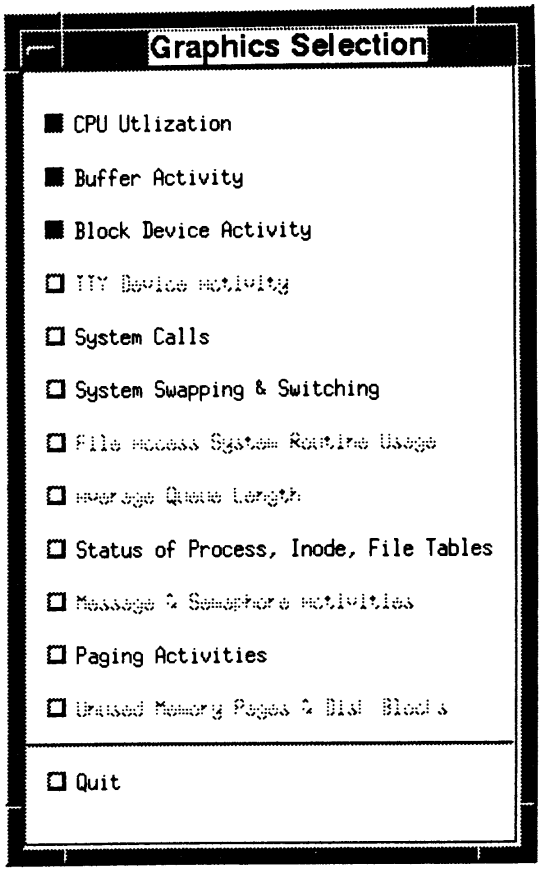


Figure 19. Pressed Graphics Selection buttons

In this CPU Utilization window, the title bar indicates the function of the drawing window. On the left of the window is the menu pane, which shows five entries: Quit, Save, Help, Pie, and Bar. At the bottom is the legend for the bar and pie charts. Above the legend is the scrolled drawing window. The label above the drawing window shows the host name as okstate, and the date of performance collection as 11/18/92. The window remains blank until the user clicks one of the drawing buttons, namely Pie or Bar, which is the indirect approach to drawing as discussed in Section 3.3.3.5.

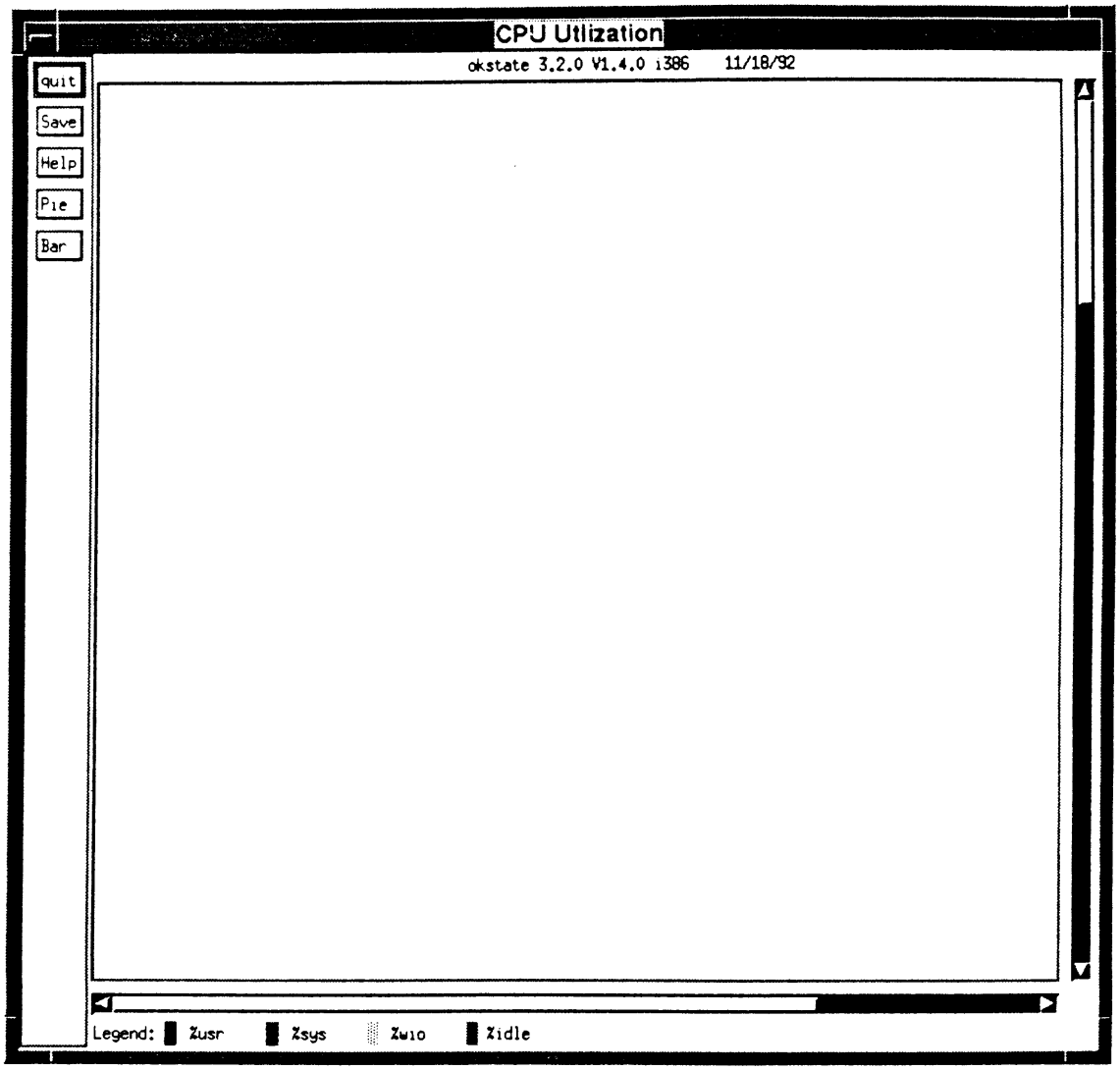


Figure 20. CPU Utilization window

Figure 21 shows the result of pressing the Pie button. The pie chart shown in Figure 21 displays the system's CPU utilization in terms of %usr, %sys, %wio, and %idle. Time is shown on top of each pie and the percentage for each item is shown on the left of each pie. The window can be enlarged to fill the screen. In case the pie chart

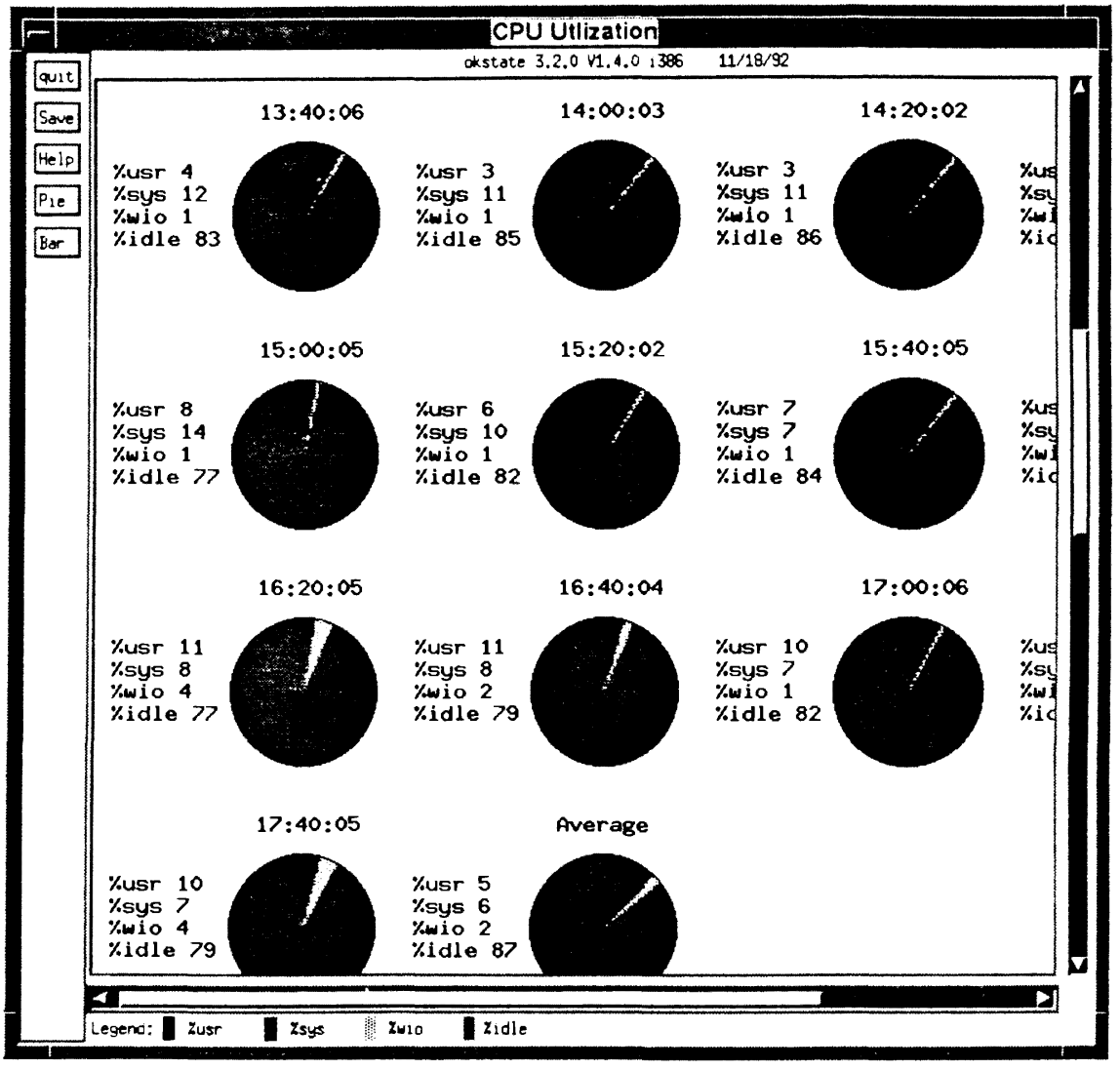


Figure 21. Pie chart of CPU utilization

is larger than the display window, the scroll bar can be slid to view images beyond the window, making it possible for a graph larger than the window (as a result of a large amount of data) to be viewable in one window instead of in multiple windows.

Pressing the help button in this display window pops up a message box, which provides a brief explanation about the four data items graphed (see Figure 22).

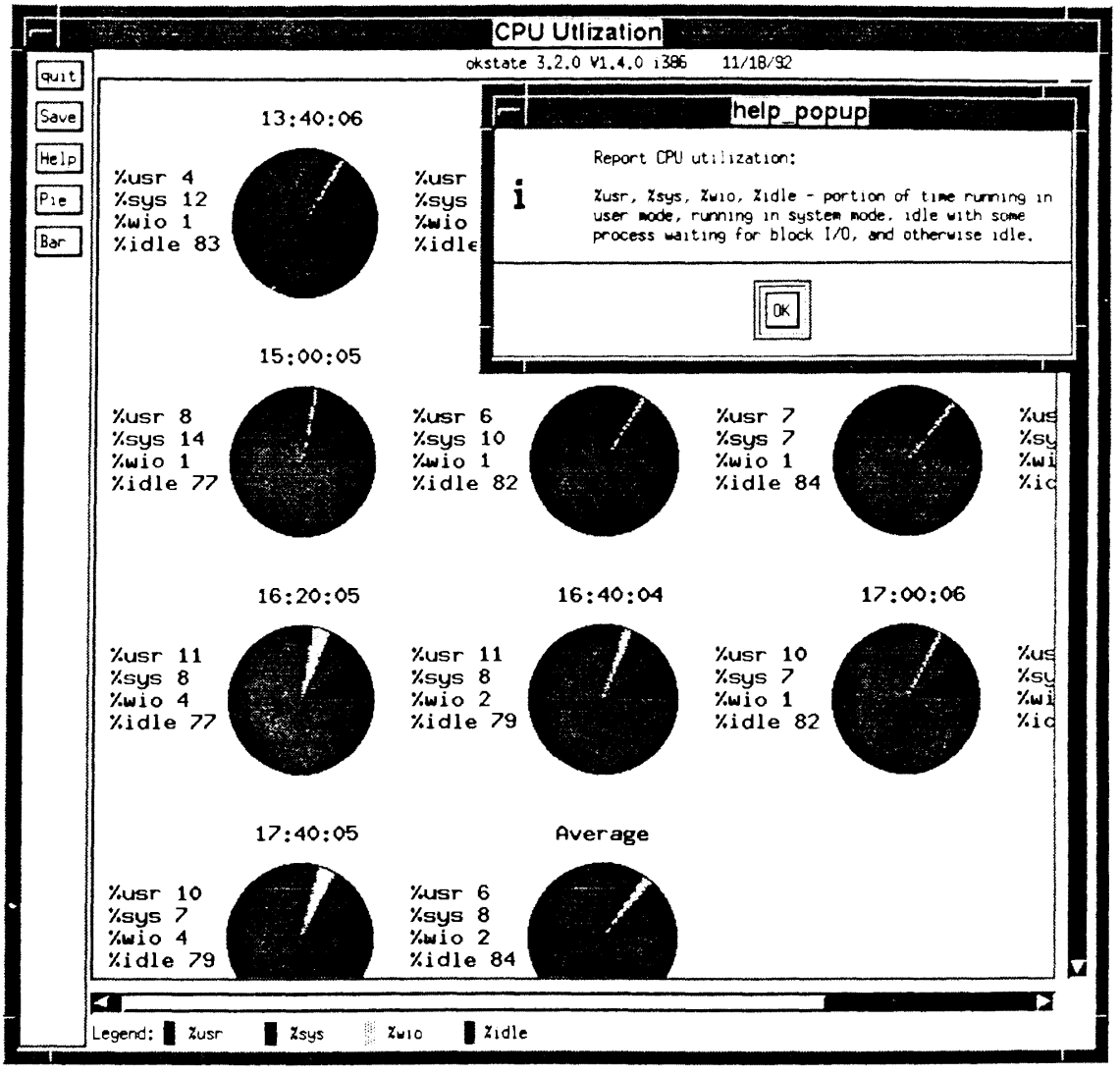


Figure 22. Help window

Clicking the Bar button shows the bar charts of the data (see Figure 23). Bar charts are basically an alternative to pie charts. As in a pie chart, the scroll bar can be used to view the part beyond the screen. The bar chart titled "%usr vs. %sys" reflects the user's utilization of the CPU on the left of the bar chart and the system's on the right as

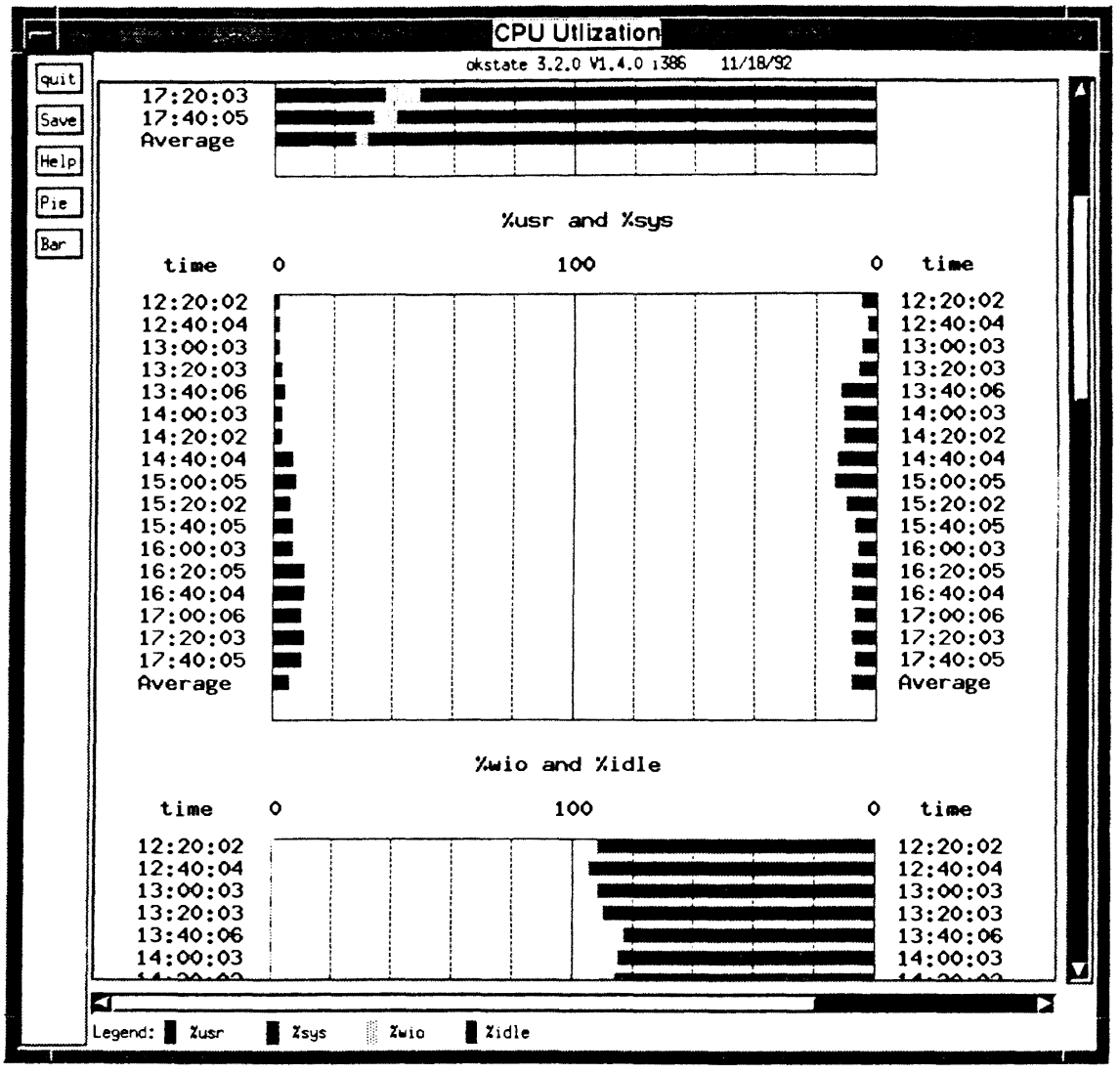


Figure 23. Bar charts of CPU utilization

a percentage. Such an arrangement is for either of two purposes: to make a comparison between the two items, if the two items are related, or to save space, if they are not related. This approach is implemented across all the bar charts of the display windows.

When the quit button of a drawing window is clicked, the window is popped down and the CPU Utilization button in the Graphics Selection window is toggled back.

Figure 24 shows the result when the button Buffer Activity in the Graphics Selection window is clicked. The popped up window has already drawn the bar charts. This is the direct approach to drawing. The menu entries only read Quit, Save, and Help. When the Save button in the drawing window is clicked, a prompt box is popped up. The prompt window prompts the user for a file name into which the specified window image is to be dumped (see Figure 25). The saved file can subsequently be viewed using the **xwud** command.

Figure 26 shows the result when the button Block Device Activity is pressed in the Graphics Selection box. This also utilizes an indirect drawing approach. The menu entries list a number of buttons indicating the block devices in addition to the common buttons. Clicking any of the device buttons will result in the display of bar charts related to that particular device's activities. Figure 27 shows the bar charts as a result of clicking the zd6 device button. Capturing of the block device activity seems to be dependent on the dynamic behavior of the system in terms of number and nature of the devices involved (a full investigation and depiction of block device activity is relegated to the

future work). As the total number of the devices changes from file to file, the buttons representing them are also set dynamically (accomplished by invoking a routine called change_device_menu).

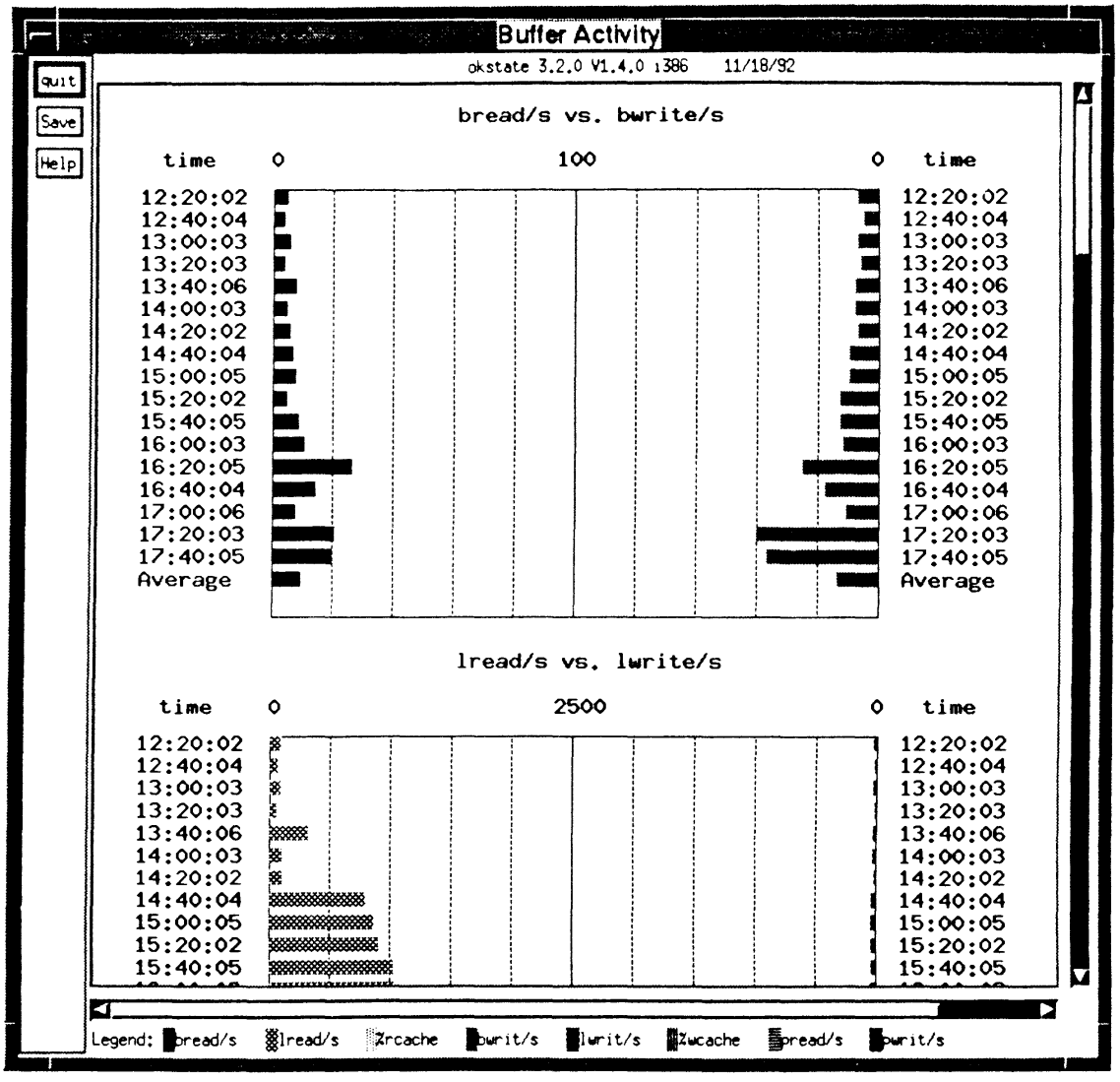


Figure 24. Buffer Activity window

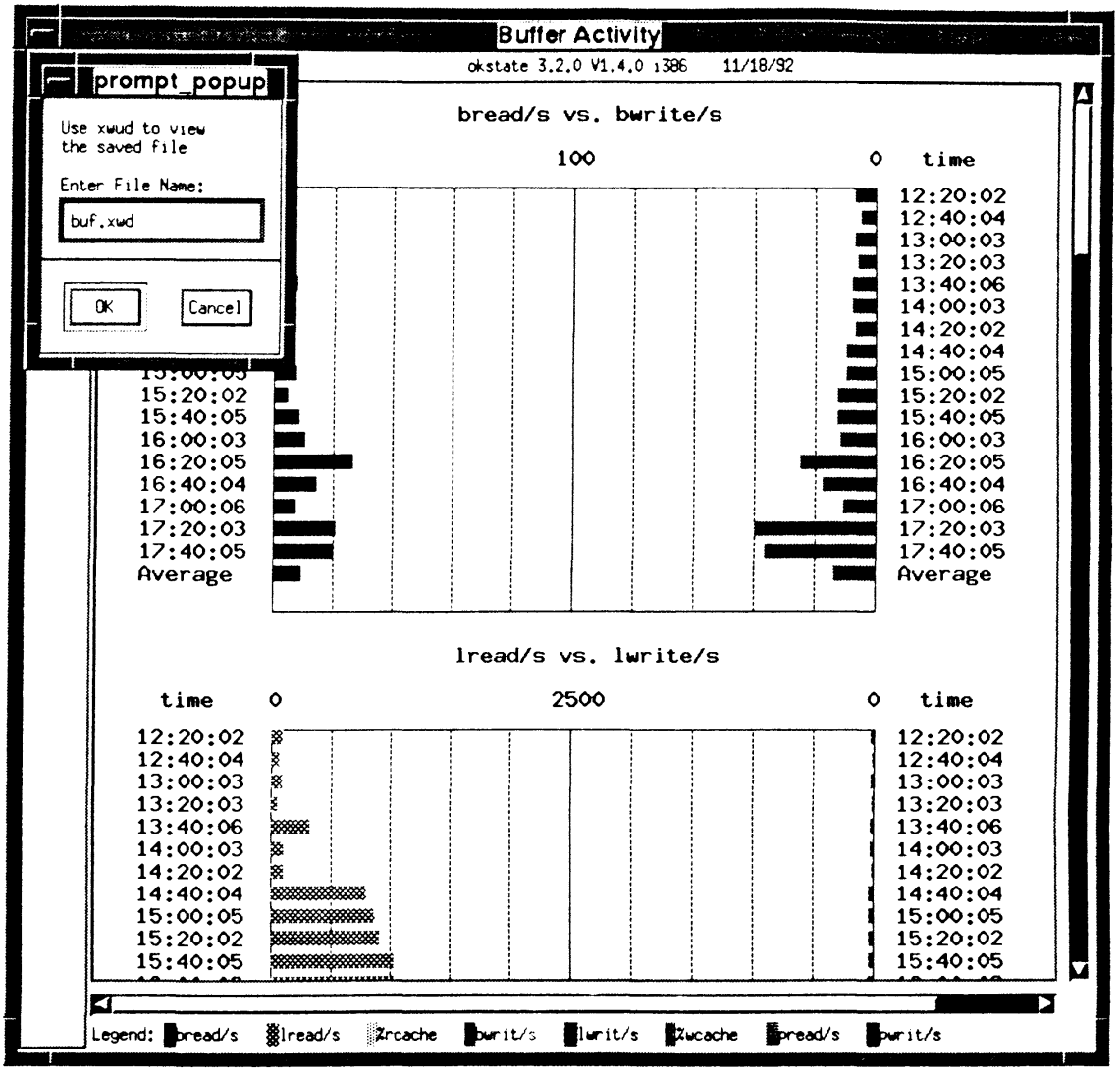


Figure 25. Prompt box for window dumping

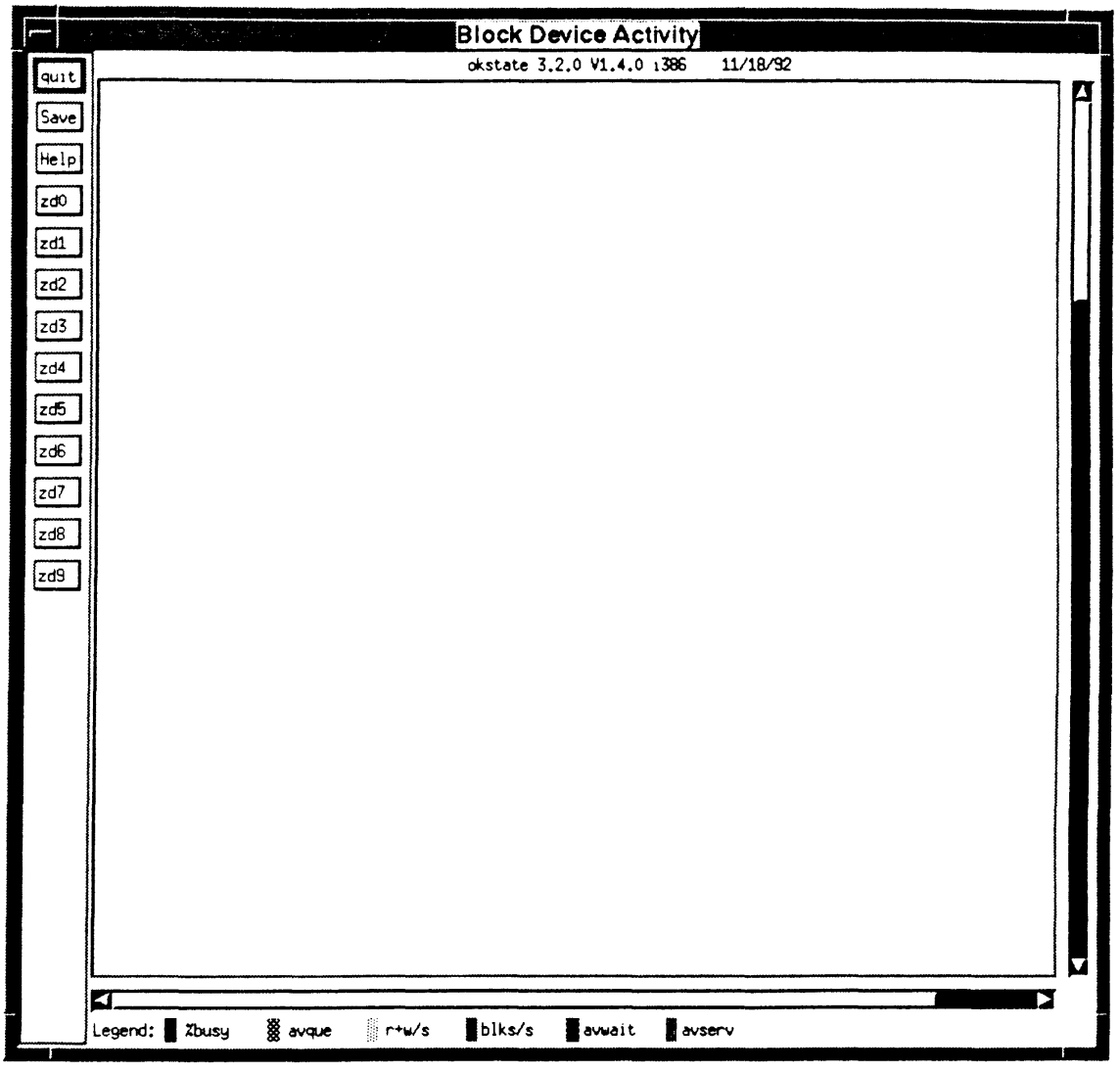


Figure 26. Blank Block Device Activity window

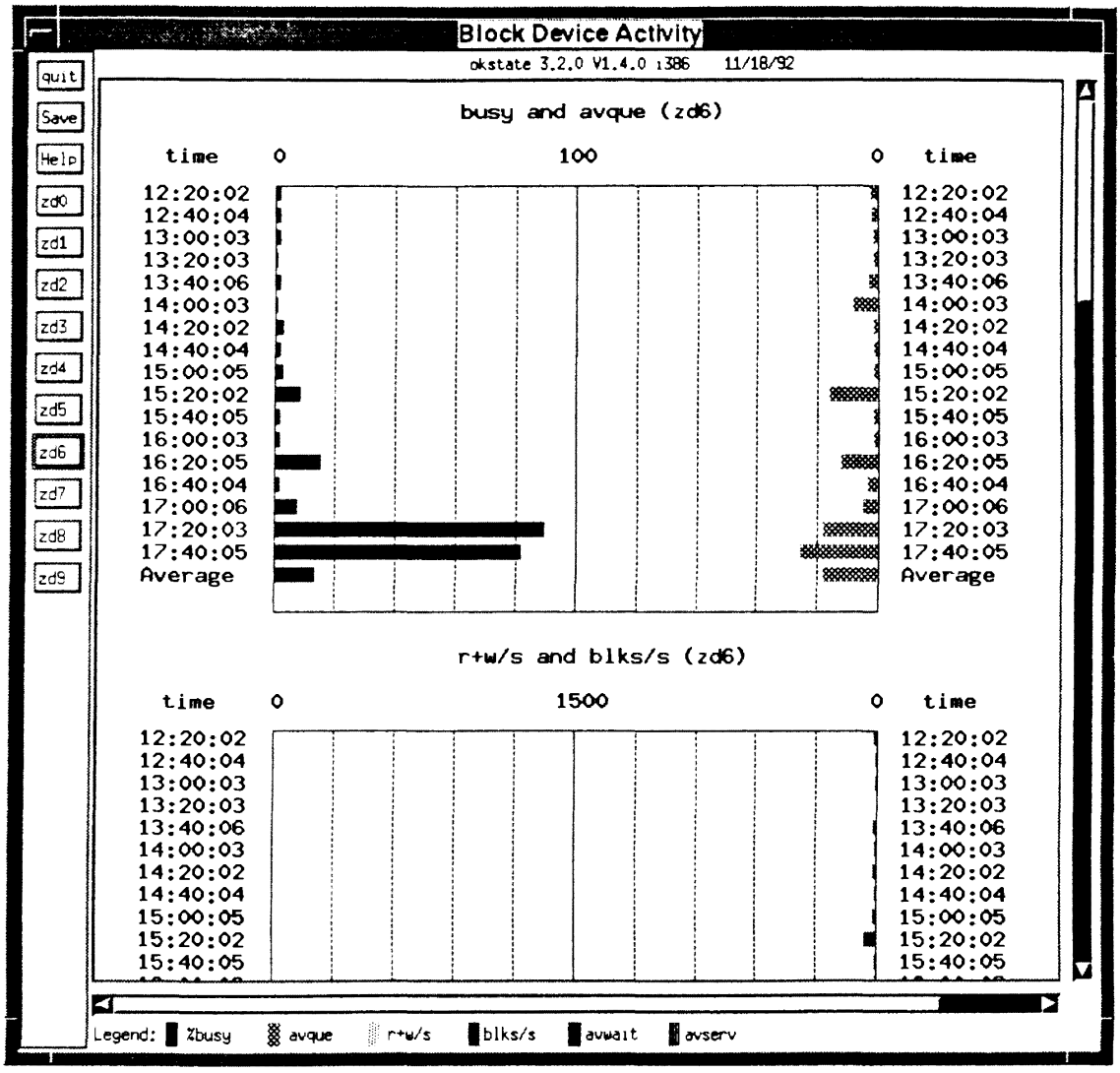


Figure 27. Drawn Block Device Activity window

CHAPTER IV

SUMMARY AND FUTURE WORK

4.1 Summary

In Chapter I, the importance of graphical user interfaces and the main goal of this thesis was stated. Chapter II covered a number of topics as background and context for the implementation work presented in this thesis. The topics covered include: a brief introduction to the Sequent S/81 machine including its architecture and operating systems; the Sequent system activity report package including commands **sar**, **sadc**, **sa1**, **sa2**, and **sag**; the subject of Graphical User Interfaces including their definition and components; the X Window System including its definition, brief history, fundamental components, client/server interaction, and software layers; OSF/Motif Programming (the most relevant part to the user interface work of this thesis) including its architecture, widget set, and programming structure. Chapter III addressed the implementation part of this thesis. Section 1 discussed the purpose and structure of the program, section 2 described the data access layer, and section 3 delineated the user interface part.

The main goal of this work was to provide a friendly user interface for display of Sequent S/81 system performance. The performance display program **sarview** was designed and implemented to achieve this goal through a number of interface objects, as

shown in Chapter III. The performance display program was shown to a number of heavy users of the Sequent system including a system administrator. It was generally agreed that the **sarview** program was user-friendly, although there is still room for improvement. Of course, this in no way constitutes a true experimental design case; rather it is considered an anecdotal evidence. A user will find that the performance display program is easier, simpler, and generally more comfortable to use in order to view and comprehend the Sequent S/81 system performance data than the **sar** or **sag** command.

The performance display program **sarview** is coded in the C language. It has about 5000 lines of code. Its main() has 57 lines of code. The total number of program procedures are up to 77 and the X routines total 60. Among the X functions, 18 are routines of Xlib, 23 are of X Toolkit, and 19 are of Motif. The libraries linked against in this program include Xlib, X Toolkit intrinsics, OSF/Motif widgets, Xmu (miscellaneous utilities), Xext (X extension library), libinet and libnsl (networking libraries), and libseq (Sequent-specific I/O library) [Sequent 90a].

4.2 Future Work

The work for future exploration may include the following.

- Enhancement of the **sarview** program to reflect the block device activity more accurately.
- Dynamic (rather than static) graphical display of various system performance measurements (analogous to the **xload** program [Sequent 91a]).
- Static or dynamic display of system accounting information.
- Statistical analysis of the collected data and displaying of the results.

- Incorporation of the performance display program into an existing simulation package (used for prototyping and evaluation of architectures and operating systems) [Daily 93] [Jhun 92] [Hassan 92] to monitor the performance of simulated systems.

REFERENCES

[Barkakati 91] Barkakati, Nabajyoti, X Window System™ Programming, Macmillan Computer Publishing, Carmel, IN, 1991.

[Berlage 91] Berlage, Thomas, OSF/Motif: Concepts and Programming, Addison-Wesley Publishing Company, Reading, MA, 1991.

[Chandrashekar 91] Chandrashekar, S., Mayfield, B. E., and Samadzadeh, Mansur H., "Project Engineering Tool to Assist in the Development and Maintenance of Project Life Cycles," Proceedings of the ACM/IEEE-CS 1991 Symposium on Applied Computing, Edited by: V. Kumar and E. A. Unger, Kansas City, Missouri, pp. 119-122, April 1991.

[Chandrashekar 93] Chandrashekar, S., Mayfield, B., and Samadzadeh, M., "Towards Automating Software Project Management," accepted for publication in The Int. Journal of Project Management, scheduled to be published in 1993.

[Daily 93] Daily, S. R. and Samadzadeh, Mansur H., "Object-Oriented Simulation of Capability Based Architectures," accepted for presentation and publication in The Twenty Sixth Annual Simulation Symposium, Sponsored by SCS, IEEE-CS, and ACM, in conjunction with The 1993 Simulation Multi-Conference, Washington D.C., March 29 - April 1, 1993.

[Hassan 92] Hassan, Khaled M. and Samadzadeh, Mansur H., "An Object-Oriented Environment for Simulation and Evaluation of Architectures," Proceedings of the IEEE 25th Annual Simulation Symposium in conjunction with The 1992 SCS Simulation Multiconference, Orlando, Florida, pp. 91-97, April 1992.

[Jhun 92] Jhun, Ik-Jeong, Hassan, Khaled M. and Samadzadeh, Mansur H., "Simulation of a Computing Environment Using Stochastic Processes and the Object-Oriented Technology," Proceedings of the Twenty-Third Annual Pittsburgh Conference on Modeling and Simulation, Volume 23, Part 3, Edited by: William G. Vogt and Marlin H. Mickle, Pittsburgh, Pennsylvania, pp. 1579-1585, April 30-May 1, 1992.

[Kobara 91] Kobara, Shiz, Visual Design with OSF/Motif, Addison-Wesley Publishing Company, Reading, MA, 1991.

[Mansfield 91] Mansfield, Niall, The X Window System: A User's Guide, Addison-Wesley Publishing Company, Reading, MA, 1991.

[Nye 90] Nye, Andrian, X Protocol Reference Manual for Version 11 of the X Window System, O'Reilly & Associates, Inc., Sebastopol, CA, 1990.

[Perkin-Elmer 85] The Perkin-Elmer Corporation, XELOS™ Administrator Guide, Oceanport, NJ, 1985.

[Rost 90] Rost, Randi J., X and Motif Quick Reference Guide, Digital Press, Burlinton, MA, 1990.

[Scheifler 92] Scheifler, Robert W. and Gettys, James, X Window System, Digital Press, Burlington, MA, 1992.

[Sequent 91a] Sequent Computer Systems, Inc., DYNIX/ptx Reference Manual, Beaverton, OR, 1991.

[Sequent 90a] Sequent Computer Systems, Inc., ptx/WINDOWS Programmer's Reference, Beaverton, OR, 1990.

[Sequent 90b] Sequent Computer Systems, Inc., ptx/WINDOWS User's Reference, Beaverton, OR, 1990.

[Sequent 90c] Sequent Computer Systems, Inc., Symmetry Multiprocessor Architecture Overview, Beaverton, OR, 1990.

[Shneiderman 87] Shneiderman, Ben, Designing the User Interface: Strategies for Effective Human-Computer Interaction, Addison-Wesley Publishing Company, Reading, MA, 1987.

[Smith 91] Smith, Jerry D., Object-oriented Programming with the X Window System Toolkits, John Wiley & Sons, Inc., New York, NY, 1991.

[Thobani 92] Thobani , A. A. and Samadzadeh, Mansur H., "Scheduling of Independent Tasks on Sequent Symmetry S/81," Proceedings of the Fifth Annual Conference: Sequent Users' Resource Forum (SURF'92), pp. 217-235, Atlanta, Georgia, October 1992.

SELECTED BIBLIOGRAPHY

Arick, Martin R., UNIX C SHELL Desk Reference, QED Information Sciences, Inc., Wellesley, MA, 1992.

Heller, Dan, Motif Programming Manual, O'Reilly & Associates, Inc., Sebastopol, CA, 1991.

Jones, Oliver, Introduction to the X Window System, Prentice-Hall, Englewood Cliffs, NJ, 1989.

Open Software Foundation, OSF/Motif Programmer's Guide, Prentice-Hall, Englewood Cliffs, NJ, 1990.

Open Software Foundation, OSF/Motif Programmer's Reference, Prentice-Hall, Englewood Cliffs, NJ, 1990.

Open Software Foundation, OSF/Motif Style Guide, Prentice-Hall, Englewood Cliffs, NJ, 1990.

Sequent Computer Systems, Inc., ptx/TCP/IP™ and Applications Overview, Beaverton, OR, 1991.

Young, Douglas, X Window System: Programming and Applications With Xt, OSF/Motif Edition, Prentice-Hall, Englewood Cliffs, NJ, 1990.

APPENDICES

APPENDIX A

GLOSSARY AND TRADEMARK INFORMATION

The following glossary has been compiled from [Nye 90], [Berlage 91], and [Kobara 91].

abstract data type: A collection of values and a set of operations on those values.

accelerator: A key combination used to invoke a menu entry quickly without displaying that menu.

application programers' interface (API): The operations a programmer can use to develop motif applications.

application window: The window where an application resides with its complete user interface.

atom: A number uniquely identifying a string inside the X server for communication.

attachment: A constraint applied to a child widget of a form widget to make one side stay in a fixed relation to some other point or widget.

background: The flat area on which a widget resides.

background color: The color from which all widgets generate their top and bottom shadows and their select color, and against which labels and bitmaps are created with the foreground color.

backing store: When a server maintains the contents of a window, the pixels saved offscreen are referred to as a backing store.

bitmap: An image created by using only two colors of the screen.

button: Either a physical button on the mouse or a widget that simulates a real button on the screen.

BSD: Berkeley Standard Distribution.

callback: A mechanism invoking a procedure of an application by a widget under specific conditions.

callback list: list of procedures to be called when a callback is activated.

child: A hierarchical reference to widgets. A child widget can be either a primitive or a manager widget. A child widget requires a parent widget to control it.

class: Common description for a set of similar objects with the same structures but different attribute values.

class hierarchy: Logical ordering of classes, in which a subclass is the specialization of its superclass. Subclasses may inherit, change, or add features.

click: Pressing and releasing a mouse button without intervening pointer motion.

client: Any widget or a set of widgets that is managed to be displayed in a window.

client-server model: A model where one process (the server) provides services to other processes (clients). Both communicate over some protocol and may reside on different machines in a network.

composite widget: Widget that may have other widgets as children and control their geometries.

common user access (CUA): IBM's prescribed graphical user interface behavior guidelines for computer systems.

common X interface (CXI): The original name of the three-dimensional widgets before they were adopted as OSF/Motif.

convenience function: A Motif-defined function that creates new widget functionality not implemented as a real widget class, rather as a creation procedure which constructs a widget composition with preset resource fields and connected by callbacks.

DECnet: A networking protocol used by Digital Equipment Corporation's VAX/VMS machines for data transfer between a server and its clients.

decoration: Area around an application window managed by the window manager to invoke window manager functions.

dialog box: A subwindow that is popped up by the application when detailed or overly abundant information needs displaying or acquiring.

event: Message sent by the X server to an application.

event-driven: A program structure in which incoming events dominate the flow of control.

event handler: A procedure to respond to one or more types of events for a specific widget.

explicit: A mode in which the mouse must be pressed in an application window so that the window can recognize that it has the keyboard input focus to the exclusion of other windows. As a result, the application will have the active input focus (also see **implicit**).

font: A complete combination of all of the characters of one size of one typeface including capitals, lowercase characters, figures, punctuation marks, and reference marks.

foreground: The name referring to the color of the components, such as labels and pixmaps, that are usually placed against the background color.

gadget: A widget without a private X window that depends on its parent widget for event dispatching, etc.

geometry management: The procedure of automatically negotiating size and location of widgets in a hierarchy.

graphics: A user command in DYNIX/ptx to provide access to graphical and numerical commands.

graphics context (GC): Various information for graphics output such as foreground pixel, background pixel, line width, clipping region, and so on. A graphics context can only be used with drawables that have the same root and the same depth as the graphics context.

graphical user interface (GUI): A visual representation of a computer's functions that can be manipulated by nonprogrammatic means.

icon: A graphic image that identifies either an application that has been miniaturized and is running or a file that is closed and not running in a file manager window.

implicit: A mode in which a window knows that it has the keyboard input focus to the exclusion of other windows as soon as the mouse is moved into the window; the mouse button does not have to be pressed (also see **explicit**).

inactive input focus: The state of a displayed window that does not have the keyboard input focus to the exclusion of other windows.

inheritance: A mechanism to make use of the functionality of a superclass in a subclass without duplicating it.

insensitive: A state in which a widget does not have any useful function in a given functional scenario. An insensitive widget's Label will be filled with a dithered pattern and will look grayed out.

inter-client communication conventions (ICCC): A set of standard rules that control interaction among clients and between a client and the window manager.

interface: The layer of graphically displayed information between a user and the computer.

intrinsics: Basic library underlying Motif that implements fundamental mechanisms for setting up widget classes.

invert: To reverse two colors, usually the foreground and background, in order to highlight a selected widget.

label: The text or pixmap placed in a widget that identifies the widget's function. The term **label** is also used for the words used to describe an area or function.

map: To make a window appear on the screen.

maximize: The action that makes a window expand and fill the entire screen.

minimize: The action that makes a window miniaturize into an icon while continuing to run.

mnemonic: Character abbreviation used for a menu command.

modal: Having multiple states in an application where a different and limited set of actions is possible.

monochrome: A video terminal that shows images in only one hue and its derivative values. Usually black and white.

Motif convenience function: A procedure specified by the Motif widgets to create dialogue type boxes such as error message boxes and information boxes.

MULTIBUS: An industry-standard bus that can be used to connect a variety of peripheral devices to an S/81 system.

palette: A set of visually compatible colors.

parent: A hierarchical term referring to widgets that can contain and have responsibility

for other primitive or manager widgets.

pixel: A single identifiable spot on the screen or in a pixmap. A pixel may have a number of different color values (black and white on a monochrome screen).

pixel values: A pixel is an N-bit value, where N is the number of bit planes used in a particular window or pixmap; that is, N is the depth of the window or pixmap. For a window, a pixel value indexes a colormap to derive an actual color to be visible.

pixmap: A three-dimensional array of bits. A pixmap is usually considered as a two-dimensional array of pixels, where each pixel can be a value from 0 to 2^N-1 and where N is the depth (z axis) of the pixmap. A pixmap can also be considered as a stack of N bitmaps.

platform: An association of hardware, operating system, peripherals, and library software that identifies a distinct computing environment.

pointer: An alternate term used to describe a mouse cursor.

POSIX: Portable Operating System Interface for Computer Environments.

primitive widget: a widget without children. Its appearance is created using basic X drawing mechanisms.

protocol: An agreement between a server and its client on how they will exchange information and how that information will be interpreted.

radio box: A set of toggle buttons, of which only one may be on at one time.

realize: Procedure of creating the X window for a widget.

requests: Commands that are sent from an application to the server demanding a specific function or attribute.

resource: entity residing in the X server and referenced through a symbolic ID (window, pixmap, graphics context, font). Also used to denote values for widget attributes.

resource database: A simple text file in which the user can specify the value of various parameters using a well-defined format.

root: A pixmap, colormap, or graphics context that, when created, shares the root of whatever drawable was used. The root of a window is the root window under which the window was created.

root window: Window representing the screen background and parent of all other windows.

server: The server offers the basic windowing mechanism. It handles inter-process communication (IPC) connections from clients, demultiplexes graphics requests onto the screens, and multiplexes input back to the appropriate clients.

shadows: The light and dark patterns surrounding most widgets to simulate a three-dimensional appearance.

shell widget: a widget governing a top-level window. It has no visible appearance but only controls the interaction with the window manager.

style guide: Rules defining a uniform behavior of Motif applications.

TCP/IP: The transmission-control protocol (TCP) and the internet protocol (IP). Two primary components in a set of protocols that permit data transferring between a server and its clients on the same or different networks.

tile: A pixmap can be replicated in two dimensions to tile a region. The pixmap itself is also referred to as a tile.

toolkit: Objects and functions available to an application programmer.

top-level window: Window that is conceptually a direct child of the root window, but may be reparented by the window manager.

top shadow: The lightest color of a widget, obtained from the background color.

tplot: A user command in DYNIX/ptx to provide graphics filters.

traversal highlight: A visual highlight that appears on or around most widgets to indicate that a particular widget has the mouse or keyboard focus to the exclusion of other widgets in the window.

traverse: The action of moving from widget to widget via the keyboard or with the mouse.

unarmed: The visual state of a widget that is unselected.

user area: Any area within an application window or a dialog box where the user can manipulate or edit data.

viewable: A window is viewable if it and all of its ancestors are mapped. This does not imply that any portion of the window is actually visible. Graphics requests can be performed on a window when it is not viewable, but output will not be kept unless the server is maintaining backing store.

visible: A region of a window is visible if someone looking at the screen can actually see

it; that is, the window is viewable and the region is not occluded by any other window.

widget: Basic interface object of the X Toolkit Intrinsics associated with an X window with encapsulated functionality.

window: A rectangular area on a display screen that is associated with a particular program.

window manager: The underlying application that makes it possible for windows to be displayed and directly manipulated on the screen.

X: A portable, networked, and transparent window system.

X client: Application process that uses the services of the X server for input and output.

Xlib: Basic C library to support the services defined in the X protocol.

Xm: The prefix Xm is placed before any value that is specified as a widget resource. A convention that identifies the X window system and Motif values as different from other values used in source code.

XmN: The prefix XmN is placed before any resource classification that asks for a specified value.

X protocol: Protocol by which X clients and X server communicate.

X server: Process that exclusively controls the display hardware and accepts requests from clients.

X terminal: A hardware device that includes the X server, but may not run clients. The clients must reside on other machines in the network.

X user interface: An intrinsics-based tool kit developed by Digital Equipment Corporation.

X toolkit intrinsics: A library of utility functions and data structures layered between Xlib and the widget set.

X Window System: Hardware-independent and network-transparent base layer that provides services to graphical user interfaces.

Xt intrinsics: See **X toolkit intrinsics**.

XY format: The data for a pixmap is in XY format if it is organized as a set of bitmaps defining individual bit planes, with the planes appearing from most significant to

least significant in bit order.

Z format: The data for a pixmap is in Z format if it is organized as a set of pixel values in scanline order.

TRADEMARK INFORMATION

DEC is a registered trademark of Digital Equipment Corporation.

DECnet, VAX, and **VMS** are trademarks of Digital Equipment Corporation.

DYNIX, DYNIX/ptx, Sequent, and **Symmetry** are registered trademarks of Sequent Computer Systems, Inc.

Hewlett Packard and **HP** are trademarks of Hewlett Packard Company.

i386 and **i486** are trademarks of Intel Corporation.

IBM, IBM PC, IBM XT, and **IBM AT** are registered trademarks of the International Business Machines Corporation.

Macintosh is a registered trademark of Apple Computers, Inc.

Microsoft, Microsoft Windows, and **MS-DOS** are registered trademarks of Microsoft Corporation.

Motif, OSF, and **OSF/Motif** are trademarks of Open Software Foundation, Inc.

Open Look is a registered trademark of AT&T Laboratories.

Presentation Manager is a trademark of IBM.

Tektronix and **Tektronix 4010** are registered trademarks of Tektronix, Inc.

The X Window System is a trademark of the Massachusetts Institute of Technology.

UNIX is a registered trademark of AT&T Laboratories.

XELOS is a trademark of Perkin-Elmer Corporation.

XUI is a trademark of Digital Equipment Corporation.

APPENDIX B

SYSTEM COUNTERS

The system activity counters sampled by the **sar** command are described below [Perkin-Elmer 85].

CPU time counters: Idle, user, kernel, and wait for I/O are the four time counters that may be incremented at each clock interrupt of 100 to 120 times per second (depending on the clock frequency). According to the mode that the CPU is in at the time of an interrupt, exactly one of the four counters is incremented.

lread and lwrite: These counters are used to record the number of logical read and write requests issued by the system to the block devices.

bread and bwrite: The bread and bwrite counters are used to record the number of times data is transferred between the system buffers and the block devices. These actual I/O operations are initiated by logical I/O operations that cannot be satisfied by the current contents of the buffers. A common measure of effectiveness of the system buffering is the ratio of the number of block I/O to logical I/O.

phread and phwrite: These tow counters are used to record the number of read and write requests issued by the system to raw devices.

swapin and swapout: The swapin and swapout counters are increased for each system request triggering a transfer from or to the swap device. Bringing a process into or out of memory usually involves more than one request because text and data are handled separately. Frequently used programs are kept on the swap device and are swapped instead of loaded in from the file system. The swapin counter captures the number of these initial loading operations and the number of resumptions of activity as well. The swapout counter reflects the degree of the actual "swapping" operations. The amount of data transferred between the swap device and memory are measured in blocks and counted by bswapin and bswapout counters.

pswitch and syscall: The pswitch and syscall counters apply to the management of multiprogramming. Syscall is increased every time a system call is invoked. The counters sysread, syswrite, sysfork, and sysexec record the numbers of invocations of **read()**, **write()**, **fork()**, and **exec()** system calls, respectively. Pswitch is used to count the times the switcher was invoked, which happens when

1. A system call lead to a road block,
2. An interrupt was invoked resulting in awakening a higher priority process, or
3. A one (1) second clock interrupt is invoked.

iget, namei, and dirblk: These three counters are related to file-access operations. Namei, an operating system routine called to perform file system path searches, searches the various directory files to get the associated i-number of a file corresponding to a special path. Iget, another operating system routine called to locate the inode entry of a file (i-number), first searches the in-core inode table. If the inode entry is not found in the table, routine iget will obtain the inode from the file system where the file exists, make an entry in the in-core inode table for the file, and return a pointer to this entry. Counter iget is always greater than counter namei because calls to iget can invoked by the namei and other file access routines as well. Counter dirblk counts directory block reads issued by the system. The average path length of files can be estimated by dividing the directory blocks read by the number of namei calls.

runque, runocc, swpque, and swpocc: These counters, applied to queue activities, are implemented in the clock.c routine. At an interval of every one (1) second, the clock routine checks the process table to see whether there are any processes in core and in ready state. If so, the counter runocc is increased and the counter runque is increased by the number of such processes. The clock routine, while checking the process table, also examines whether any processes in the swap device are in ready mode. If the swap queue is occupied, the count swpocc is increased and the counter swpque is increased by the number of processes in swap queue.

readch and writech: These two counters are used to count the total bytes (characters) transferred by the **read** and **write** system calls, respectively.

monitoring terminal device activities: Six counters are used to monitor terminal device activities. Rcvint, xmint, and mdmint are used to measure hardware interrupt occurrences for receiver, transmitter, and modem, respectively. Rawch, canch, and outch are used to count the number of characters in the raw queue, canonical queue, and output queue, respectively. Characters produced from devices operating in the cooked mode, such as terminals, are counted in both rawch and (as edited) in canch; but those generated by raw devices, such as communication processors, are counted only in rawch.

msg and sema counters: The msg and sema counters are used to count message sending and receiving times and semaphore operations, respectively.

monitoring I/O activities: There are four counters kept for each disk or tape drive in the device status table. The io_ops counter is incremented when an I/O operation has been invoked on the device. This includes block I/O, swap I/O, and physical I/O. Counter io_bcnt measures the amount of data transferred between the device and memory in 512-byte units. Io_act and io_resp count the active time and response time of a device, respectively, in time ticks summed over all I/O requests that have finished for each device. The device active time includes the device seeking, rotating, and data transferring times, whereas the response time of an I/O operation is counted from the time when the I/O request is queued to the device to the time the I/O finishes.

inodeovf, fileovf, textovf, and procovf: These four counters come from the _syserr structure. When an overflow occurs in any of the inode, file, text, and process tables, the corresponding overflow counter is increased.

APPENDIX C

SYSINFO STRUCTURE

The structure of the binary daily data file is composed of one word (int) that represents the number of elements in the devio array actually written followed by zero or more copies of the sa structure. The structure is defined below [Sequent 91].

```
struct sysinfo {

  time_t cpu[5]; /* <time> type */
                 /* report CPU utilization time running in  */
                 /* user mode, running in system mode, idle */
                 /* with some process waiting for block     */
                 /* I/O, and otherwise idle.                */


#define CPU_IDLE    0
#define CPU_USER    1
#define CPU_KERNEL  2
#define CPU_WAIT    3
#define CPU_SXBRK   4

  time_t wait[3];

#define W_IO   0
#define W_SWAP 1
#define W_PIO  2

             /**** Report buffer activity ****/

  long   bread;  /* transfers (per second) of data between */
  long   bwrite; /* system buffers and disk or other block */
                 /* devices                                */

  long   lread;   /* accesses of system                    */
  long   lwrite;
```

```
    long   phread;  /* transfers via raw (physical) device    */
    long   phwrite; /* mechanism                              */

   /* **** Report system swapping and switching activity *****/

    long   swapin;  /* # of transfers for swapins             */
    long   swapout; /* # of transfers for swapouts            */
    long   bswapin; /* # of 512-byte units transferred for    */
                    /* swapins                                */

    long   bswapout; /* # of 512-byte units transferred for   */
                     /* swapouts                              */

    long   pswitch;  /* process switch                        */

    long   syscall;  /* system calls of all types             */
    long   remcall;
    long   fileop;
    long   serve;

    /* ***** Report system calls ***** */

    long   sysread;  /* specific system calls */
    long   syswrite;
    long   sysfork;
    long   sysexec;

  /*  Report average queue length while occupied,          */
  /*  and % of time occupied                                */

    long   runque;   /* run queue of processes in memory and */
    long   runocc;   /* runnable; %runocc                    */
    long   swpque;   /* swap queue of processes swapped out  */
    long   swpocc;   /* but ready to run; %swpocc            */

    /* ***** Report use of file access system routines ***** */

    long   iget;     /* apply to file-access operations      */
    long   namei;    /* file system path searches            */
    long   dirblk;   /* dirblk/namei => avg file path length */
    long   readch;   /* char transferred by read sys calls   */
    long   writech;  /* char transferred  by write sys calls */

    /* ****** Report TTY device activity ******  */

    long   rcvint;   /* receive, transmit and modem interrupt */
    long   xmtint;   /* rates                                 */
    long   mdmint;
    long   rawch; /* input ch rate (# of ch in the raw queue)*/
    long   canch; /* input ch rate processed in canonical    */
                  /* queue                                   */

    long   outch;  /* output character rate in output queue  */
```

```
  /* *** Report message and semaphore activities *** */

  long   msg;    /* primitives per second */
  long   sema;

  long   pnpfault;
  long   wrtfault;
};
struct minfo {
  ulong freemem[2]; /* freemem in pages "double" long  */
                    /* freemem[0] least significant    */

  long freeswap;    /* disk blks available for         */
                    /* process swapping                */
  long vfault;      /* address translation page faults */
                    /* (valid page not in memory)      */

  long   demand;   /*  demand zero and demand fill pages */
  long   swap;     /*  pages on swap                     */
  long   cache;    /*  pages in cache                    */
  long   file;     /*  pages on file                     */
  long   pfault;
  long   cw;
  long   steal;
  long   freedpgs;
  long   unmodsw;
  long   unmodfl;
};

struct dev_stats {
  char   d_name[12]; /* device name                          */
  ulong  d_ops;      /* number of I/O requests since boot    */
  ulong  d_blks;     /* number of DEV_BSIZE blocks xferred   */
  ulong  d_active;   /* cumulative time device was active    */
                     /* in milliseconds                      */
  ulong  d_resp;     /* cumulative response time             */
                     /*in milliseconds                       */
};

struct sa {
  struct sysinfo si;
  struct minfo mi;
  int  szinode;  /* current size of inode table             */
  int  szfile;   /* current size of file table              */
  int  szproc;   /* current size of proc table              */
  int  szlckf;   /* current size of file record header      */
                 /* table                                   */
  int  szlckr;   /* current size of file record lock table  */
  int  mszinode; /* size of inode table                     */
  int  mszfile;  /* size of file table                      */
  int  mszproc;  /* size of proc table                      */
  int  mszlckf;  /* max size of file record header table    */
  int  mszlckr;  /* maximum size of file record lock table  */
```

```
  long inodeovf; /* cumulative overflows of inode table    */
  long fileovf;  /* cumulative overflows of file table     */
  long procovf;  /* cumulative overflows of proc table     */
  time_t  ts;    /* time stamp, seconds                    */
  struct dev_stats devio[1]; /* device unit information    */
};
```

APPENDIX D

System Activity Report Command sar

The twelve subset options for sar are described below [Sequent 91].

(1) -u This is the default argument. It reports CPU utilization by four (4) items, %usr, %sys, %wio, %idle, each of which is derived from the formula:

%-of-cpu-x = cpu-x / (cpu-idle + cpu-usr +

cpu-sys + cpu-wio) * 100

where cpu-x is cpu-idle, cpu-usr, cpu-sys, or cpu-wio.

(2) -b This option reports buffer activity. Bread/s and bwrit/s refer to transferring per second of data between system buffers and disk or other block devices; lread/s and lwrit/s report accesses per second of system buffers; %rchche and %wcache reflect cache hit ratios, derived from the formula:

%-of-cache-I/O = (logical-I/O - block-I/O) /

logical-I/O * 100

where cache I/O is cache read or cache write; pread/s and pwrit/s reveal the transferring per second via raw (physical) device mechanism.

(3) -d The option reports activity for each device such as a disk or tape drive. When data is displayed, the device specification for representing a disk drive in the Sequent is zd#, and that for a tape drive is zt#, where # is a numerical value. The

activity data reported includes six items. %busy reports the potion of time the device was busy with handling a transfer request, which comes from the formula:

%-ofbusy = I/O-active / (t2 -t1) * 100

where t2 and t1 represent two distinct times at which the data was sampled; avque shows the average number of requests outstanding during that period, that is,

avg-queue-length = I/O-resp / I/O-active

The item r+w/s shows the number per second of data transfers from or to the device, while the item blks/s the number per second of bytes transferred in 512-byte units. avwait reflects the average time in millionsecond that transfer requests wait idly on queue, which is:

avg-wait = (I/O-res - I/O-active) / I/O-ops

where I/O-ops is the number of I/O operations on the device; avserv shows the average time to be serviced (the time for disks includes seek, rotational, latency and data transfer times), which comes from the formula:

agv-service-time = I/O-active / I/O-ops

(4) -y This option reports TTY device activity, which includes six items. rawch/s, canch/s, and outch/s reveal input character rate, input character rate processed by canon, and output character rate, respectively; rcvin/s, xmtin/s, and mdmin/s reflect the receive, transmit and modem interrupt rates, respectively.

(5) -c The argument selection reports system call activity. scall/s shows system calls of all types; sread/s, swrit/s, fork/s, exec/s reflect specific system calls; rchar/s and wchar/s reveal the number of characters transferred by read and write system calls.

(6) -w This option reports system swapping and switching activity. Five (5) items are

involved in reporting this activity. swpin/s and swpot/s show the number of transfers while bswin/s, and bswot/s the number of 512-byte units transferred for swapins and swapouts which include initial loading of some programs. pswch/s reflects the number of process switches.

(7) -a The selection of this option reveals the use of file access system routines which include iget/s, namei/s, and dirblk/s.

(8) -q This option reports queue activity. The item runq-sz shows the average run queue of processes in memory and runnable while occupied, derived from the formula:

avg-run-queue-length = run-queue /

run-queue-occupied-time

while %runocc reflects the percentage of time occupied, coming from the formula:

%-of-run-queue-occupied-time =

run-queue-occupied-time / (t2 - t1)

swpq-sz shows the average swap queue of processes swapped out but ready to run while occupied, derived from:

avg-swap-queue-length = swap-queue /

swap-queue-occupied-time

whereas %swpocc reflect the percentage of time occupied by the swap queue, coming from

%-of-swap-queue-occupied-time =

swap-queue-occupied-time / (t2 - t1)

(9) -v The choice of this option provides the status of process, inode, and file tables. The items text-sz, proc-sz, inod-sz, file-sz and lock-sz represent the entries/size

for each table, which was evaluated once at each sampling point; The item ov shows the overflows that occur between sampling points for each table.

(10) -m This option reports message and semaphore activities. The items msg/s and sema/s show the number of the corresponding primitives per second.

(11) -p This optional argument provides paging activity information. Four (4) items are involved in this reporting. vflt/s shows the number of address translation pagefaults (valid page but not in memory); pflt/s reveals the number of page faults from protection errors (illegal access to page) or "copy-on-writes" (always 0 under DYNIX); pgfil/s reflects the vflt/s satisfied by page-in from file system; rclm/s shows the number of valid pages reclaimed for free list (always = 0 under DYNIX).

(12) -r This option reports the unused memory pages and disk blocks. The item freemem shows the average pages available to user processes and the item freeswap reflects the disk blocks available for process swapping. The following shows some examples of how to use **sar**:

-- if you want to see today's CPU activity so far,

use **sar**

-- if you want to watch CPU activity evolve for 20 minutes and save the results,

use **sar** -o temp 60 20

where temp is the file name into which the results will be saved, 60 is the interval in seconds and 20 is the times

-- if you want to later review paging activities from that period,

use **sar** -d -f temp

APPENDIX E

MANUAL PAGE FOR THE **sarview** PROGRAM

Name

sarview - system activity graphical displayer

Synopsis

sarview

Description

The **sarview** program is an OSF/Motif-based graphical user interface tool for graphical display of system performance under DYNIX/ptx on the Sequent Symmetry S/81 system. It has been designed to use **sar** (a DYNIX/ptx user command) to generate a graphical system activity report.

The visible objects of **sarview** include a main window, three dialogue boxes, and up to twelve displaying windows, plus two message boxes and one prompt box that are popped up upon demand. The main window, titled System Activity Graphical Displayer, has a menu pane which displays the main menu entries, namely File, Options, Graphics, and Quit, and a scroll text window which provides some information on the **sarview** program.

The dialogue box, titled File Selection, provides for the selection of an **sar** data file. The default directory is set to /usr/adm/sa, where DYNIX/ptx keeps its daily file of system activity data. The Option Selection dialogue box helps select from among the twelve options about the system performance report and the specification of the start time, end time, and interval amount of the report. The dialogue box, titled Graphics Selection, provides for the selection from those items that were previously chosen and are to be displayed.

The displaying window presents the specified data about the system performance in the form of bar charts or pie charts. The bar or pie charts can be scrolled vertically or horizontally in the window if they are longer or wider than the drawing window.

One message box is used for echoing errors and the other for providing on-line help in the drawing window about the charts being displayed. The prompt box is used to prompt a user for the name of a file into which a specific drawing window image is to be dumped.

Two kinds of graphical displaying are used: direct and indirect. By direct, it is meant that the graphs show up when the display window is popped up. When it is indirect, the display window that has been popped up remains blank until the user presses one of the drawing buttons. Two of the options take the indirect approach, namely CPU Utilization and Block Device Activity. The rest of the options take the direct approach.

sarview invokes **sar** (see man page for **sar**) to get data for its graphical output. Thus, a file to be used by **sarview** must be acceptable to **sar**.

Options

As an Xt Toolkit-based program, **sarview** accepts the standard X Toolkit command line options. The commonly-used options include:

-bg color
: This option defines the color to be used for the background of the window. The default is white.

-fg color
: This option defines the color to be used for the foreground of the window. The default is black.

-name name
: This option defines the application name under which resources are to be obtained. The parameter name should not contain "." or "*" characters. The default is the executable file name.

-title string
: This option defines the window title string, which may be displayed by window managers. The default title is the command line specified after the -e option, if any; if not, the application name is used.

-rv
: This option reverses the foreground and background colors.

-geometry geometry
: This option defines the user preferred size and position of the application window.

-display display
: This option defines the X server to which the program is connected.

-iconic

This option indicates that **sarview** should ask the window manager to start it as an icon instead of as the normal window.

See Also

sar, sadc, sar1, sar2, sag

APPENDIX F

SYSTEM ADMINISTRATOR'S MANUAL FOR **sarview**

1. Description

The **sarview** program is an OSF/Motif-based graphical user interface tool for graphical display of system performance under DYNIX/ptx on the Sequent Symmetry S/81 system. It has been designed to use **sar** (a DYNIX/ptx user command) to generate a graphical system activity report.

The visible objects of **sarview** include a main window, three dialogue boxes, and up to twelve displaying windows, plus two message boxes and one prompt box that are popped up upon demand. The main window, titled System Activity Graphical Displayer, has a menu pane which displays the main menu entries, namely File, Options, Graphics, and Quit, and a scroll text window which provides some information on the **sarview** program. The contents of the scroll text window is read in from a text file named des.text.

The dialogue box titled File Selection provides for the selection of an **sar** data file. The default directory is set to /usr/adm/sa, where DYNIX/ptx keeps its daily file of system activity data. The Option Selection dialogue box provides for the selection among the twelve options (about the system performance report) and the specification of the start time, the end time, and the interval amount of the report. The dialogue box titled Graphics Selection provides for the selection from those items that were previously chosen and are to be displayed.

The displaying window presents the specified data about the system performance in the form of bar charts or pie charts. The bar or pie charts can be scrolled vertically or horizontally in the window if they are longer or wider than the drawing window.

One message box is used for echoing error and the other for providing on-line help in the drawing window about the charts being displayed. The on-line help message is read in from a text file called help.text during execution. The prompt box is used to prompt a user for the name of a file into which a specific drawing window image is to be dumped.

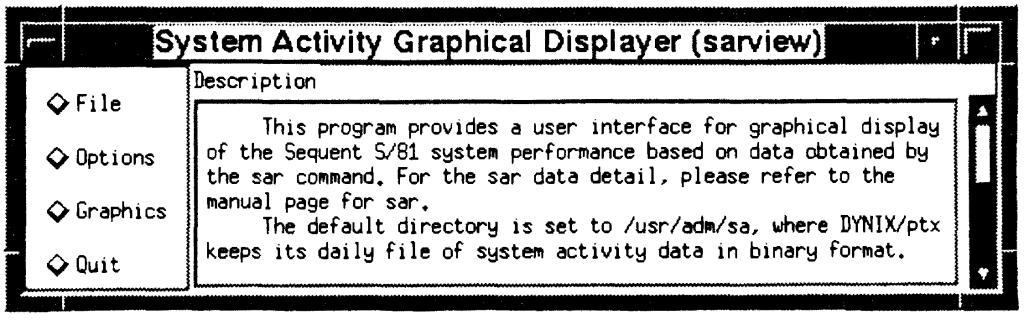
Two kinds of graphical displaying are used: direct and indirect. By direct, it is meant that the graphs show up when the display window is popped up. When it is

indirect, the display window that has been popped up remains blank until the user presses one of the drawing buttons. Two of the options take the indirect approach, namely CPU Utilization and Block Device Activity. The rest of the options take the direct approach.

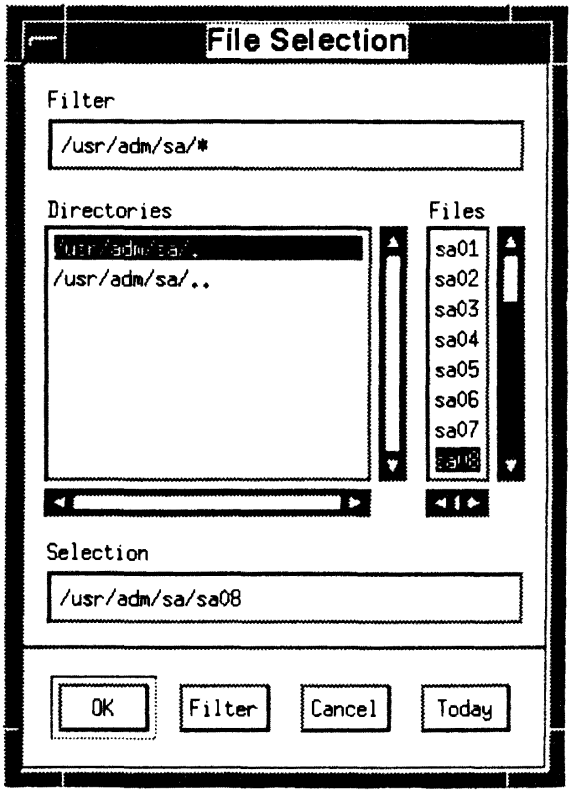
sarview invokes **sar** (see man page for **sar**) to get data for its graphical output. Thus, a file to be used by **sarview** must be acceptable to **sar**.

2. Interface objects of the **sarview** program

The main menu window looks like this:

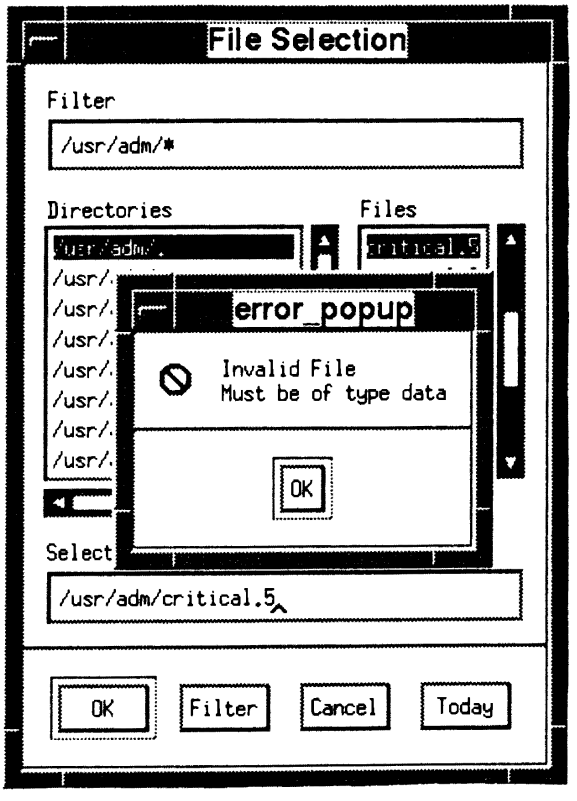


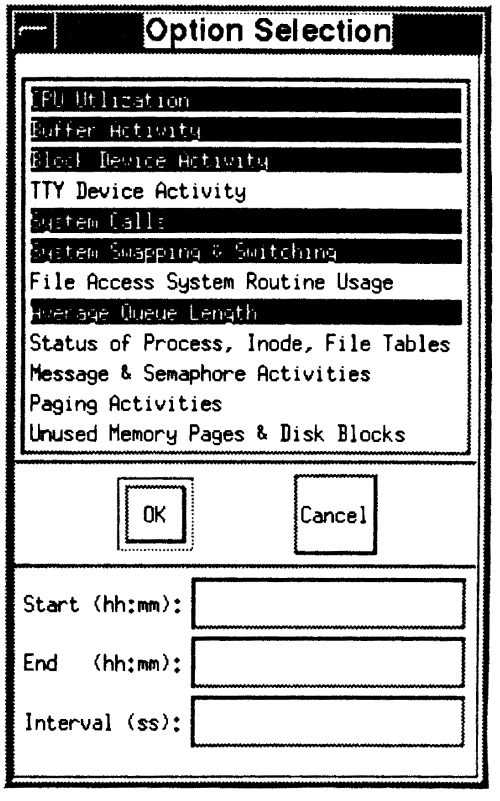
When the File button in the main menu window is pressed, a File Selection window is popped up as shown on the right. Through this File Selection window, a file of data type that was generated by running the **sar** command can be selected. Pressing the Today button selects the file of the current date. When the file selection is finished, click the OK button to pop down the window.



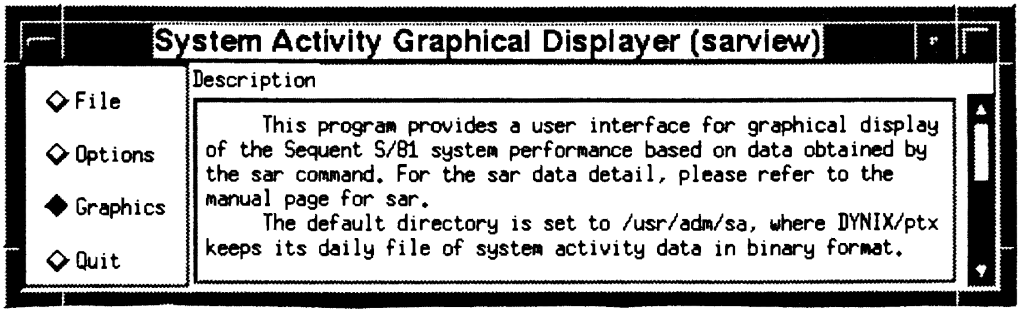
However, if the selected file is not of data type, pressing the OK button will result in popping up an error message box to echo this error (see the left figure). The user must respond to it before any further interaction can take place.

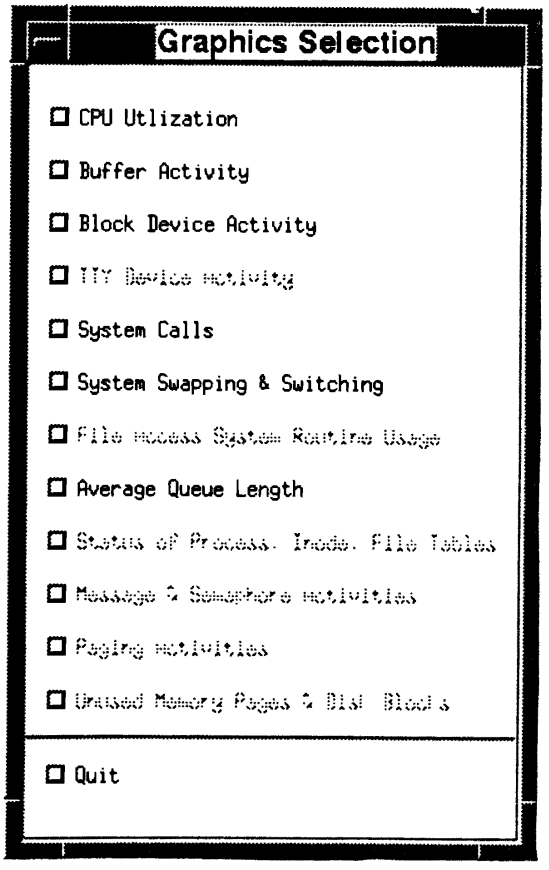
Clicking the Options button in the main menu window pops up an Option Selection window, which provides for the selection from among the twelve options about the system performance (see the right figure). The options for the start time, the end time, and the amount of the interval can be made through the selection pane at the button of this window.



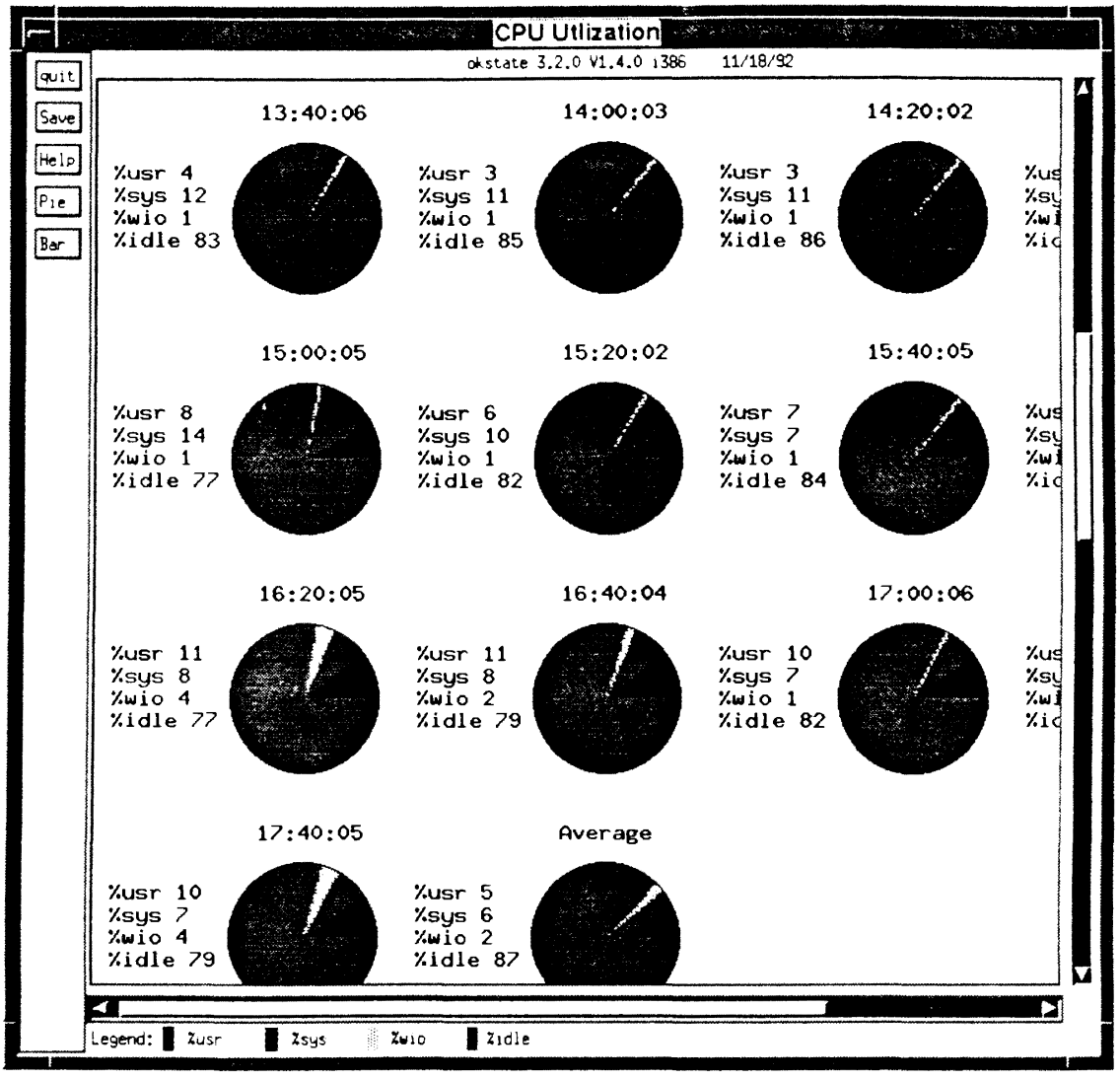


Pressing the Graphics button in the main menu window pops up a graph selection window. Those options that have not been chosen will look faded.

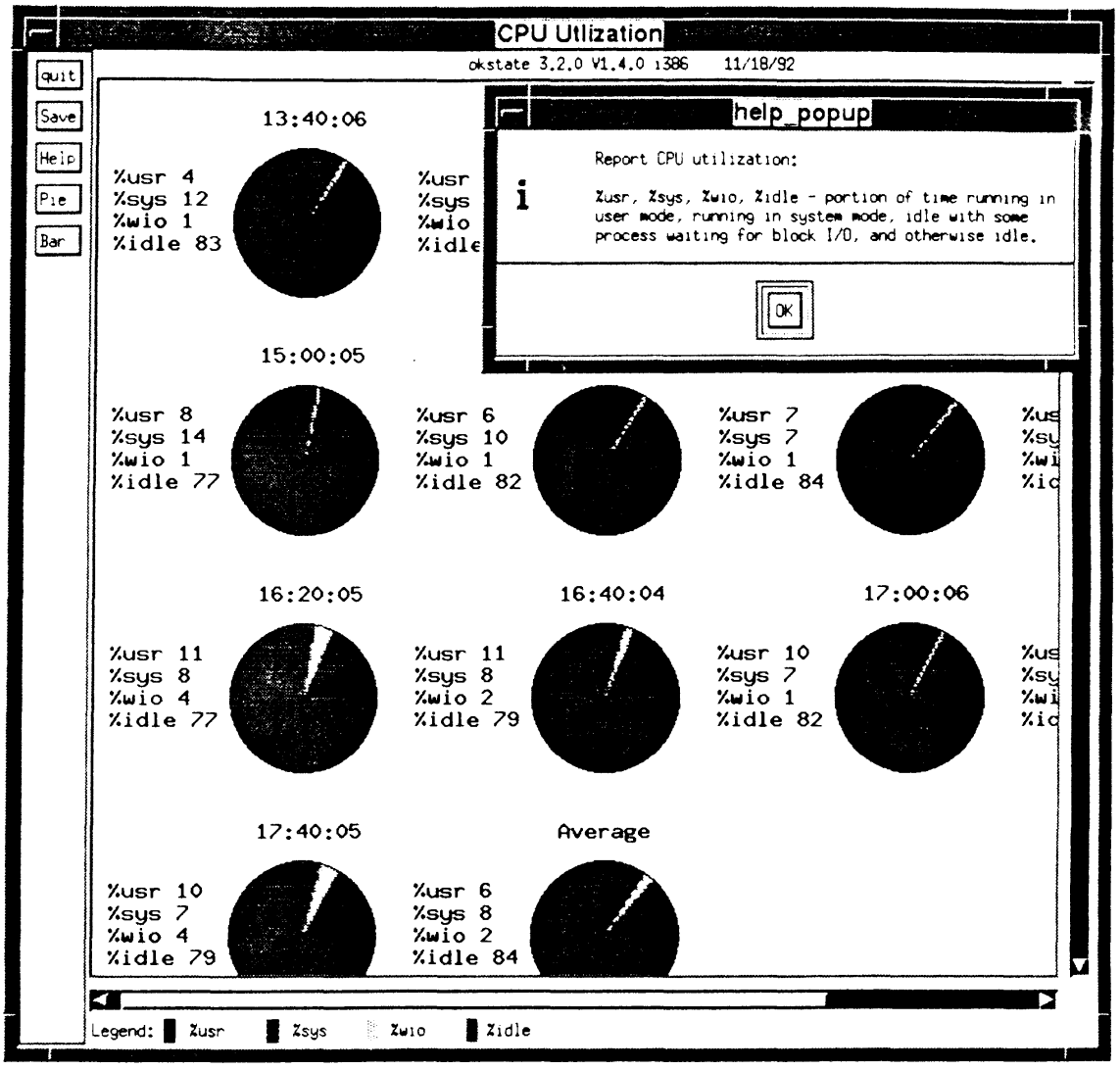




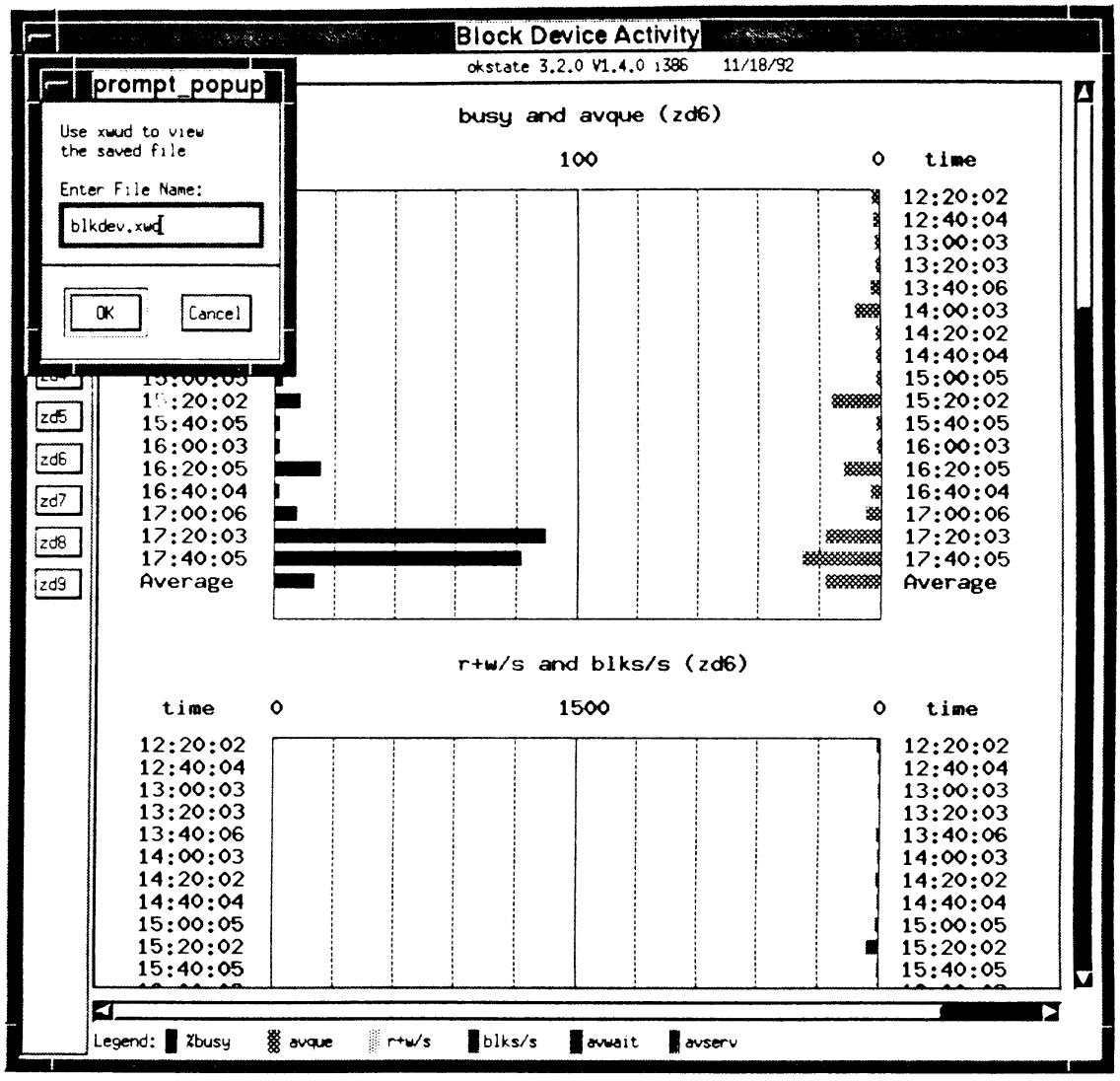
Clicking a button that has not been faded pops up a display window corresponding to the option indicated by the button. Below shows a drawing window of a pie chart of CPU utilization.



On-line help is provided through a help window by clicking the Help button in the menu pane of a display window.



Clicking the Save button in the menu pane of a display window pops up a prompt window, through which the user can specify the name of a file, into which the window image is to be dumped. The dumped image can be viewed using the **xwud** command.



3. Maintenance

To update the content of the scroll text window in the main window or that of the on-line help window, any ASCII file editor can be used to modify the des.text file or the help.text file, respectively. When editing the on-line help message, attention must be paid to the separators that are placed both at the beginning and the end of one help message in the help.text file.

The two supporting files, des.text and help.text, need to be placed in the same directory as the executable file. It is recommended that the user not be allowed to make any change to the des.text file or the help.text in order to keep the integrity of the supporting files.

To facilitate the more detailed view of system performance, the **sar** daily file under the directory /usr/adm/sa may be generated with smaller interval amounts during certain periods of time, e.g., peak times.

To add a menu button in the main menu window, the routine create_menu needs to be modified. To add an option button in other menu panes, the corresponding routines need to be modified. For example, if a button needs to be added into the Option Selection window menu pane, the routine called create_opselbox needs to be modified.

4. Options

As an Xt Toolkit-based program, **sarview** accepts the standard X Toolkit command line options. The commonly-used options include:

-bg color
: This option defines the color to be used for the background of the window. The default is white.

-fg color
: This option defines the color to be used for the foreground of the window. The default is black.

-name name
: This option defines the application name under which resources are to be obtained. The parameter name should not contain "." or "*" characters. The default is the executable file name.

-title string
: This option defines the window title string, which may be displayed by window managers. The default title is the command line specified after the -e option, if any; if not, the application name is used.

-rv

This option reverses the foreground and background colors.

-geometry geometry
This option defines the user preferred size and position of the application window.

-display display
This option defines the X server to which the program is connected.

-iconic
This option indicates that **sarview** should ask the window manager to start it as an icon instead of as the normal window.

5. Related Commands

The commands related to the **sarview** command under DYNIX/ptx include **sar**, **sadc**, **sar1**, **sar2**, and **sag**.

APPENDIX G

PROGRAM LISTING

The **sarview** program is structured as shown in Figure 5. The listing is in the following order:

sarview.h
getdata.c
motif.h
menu.h
menu.c
des.text
graph.h
graph.c
help.text

```
/**** sarview.h  ****/
/**** sarview  global variables and type declarations ****/

#ifndef H_SARVIEW
#define H_SARVIEW

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <Xm/Xm.h>
#include <X11/Shell.h>
#include <X11/Intrinsic.h>

#define LENGTH 10

/* Sa data structure */
typedef struct { /* report cpu utilization */
  char time[LENGTH];
  char usr[8]; /* portion of time runing in user mode */
  char sys[8]; /* portion of time running in system mode */
  char wio[8]; /* portion of time running in I/O mode */
  char idle[8]; /* portion of time running in idle mode */
} CPU_Utilization;

typedef struct { /* report buffer activity */
  char time[LENGTH];
  int  bread; /* block read */
  int  lread; /* logical read */
  int  rcache; /* catch hit ratio */
  int  bwrite; /* block write */
  int  lwrite; /* logical write */
  int  wcache; /* catch hit ratio */
  int  pread;  /* transfers via raw (physical) device mechanism */
  int  pwrite;
```

```
} Buffer_Activity;

typedef struct{
  char  device[4]; /* device name */
  int   busy;
  float avque;  /* average queue */
  int   r_w;    /* read and write */
  int   blks;   /* blocks */
  float avwait; /* average wait */
  float avserv; /* average service */
} Blk_Data;

typedef struct { /* report block device activity */
  char  time[LENGTH];
  Blk_Data data[15];
  int dev_no; /* the number of device */
} Block_Device;

typedef struct { /* report TTY device acitvity */
  char time[LENGTH];
  int rawch; /* input character rate (# of characters in the raw queue) */
  int canch; /* input character rate processed in canonical queue */
  int outch; /* output character rate in output queue */
  int rcvin; /* receive rate */
  int xmtin; /* transmit rate */
  int mdmin; /* modem interrupt rate */
} TTY;

typedef struct { /* report system calls */
  char time[LENGTH];
  int  scall; /* system calls of all types */
  int  sread; /* specific system calls */
  int  swrit;
  float fork;
  float exec;
  long rchar; /* characters transferred by read sytem calls */
  long wchar; /* characters transferred by write system calls */
} Sys_Call;

typedef struct {  /* report system swapping and switching activity */
  char time[LENGTH];
  float swpin; /* # of transfers for swapping in */
  float bswin; /* # of 512-byte units transferred for swapping in */
  float swpot; /* # of transfers for swapping out */
  float bswot; /* # of 512-byte units transferred for swapping out */
  int   pswch; /* process switch */
} Sys_Swap;

typedef struct { /* report file access activity */
  char time[LENGTH];
  int  iget;   /* apply to file-access operations */
  int  namei;  /* file system path searches */
  int  dirbk;  /* dirblk/namei => average file path length */
} File_Access;

typedef struct { /* report queue activity */
  char time[LENGTH];
  float runq_sz; /* run queue of processes in memory and runnable */
  int   runocc;  /* percentage of runnable processes */
  float swpq_sz; /* swap queue of processes swapped out but ready to run */
  int   swpocc;  /* percentage of swap_sz */
} Queue_Length;

typedef struct { /* report table status */
  char time[LENGTH];
  int  proc_sz;  /* current size of procedure table */
  int  proc_tsz; /* maximum size of procedure table */
  int  proc_ov;  /* cumulative overflows of procedure table */
  int  inod_sz;  /* current size of inode table */
  int  inod_tsz; /* maximum size of inode table */
  int  inod_ov;  /* cumulative overflows of inode table */
  int  file_sz;  /* current size of file table */
  int  file_tsz; /* maximum size of file table */
  int  file_ov;  /* cumulative overflows of file table */
  int  lock_sz;  /* current size of file record lock table */
  int  lock_tsz; /* maximum size of file record lock table */
} Table_Status;
```

```
typedef struct { /* report message and semaphore activities */
  char time[LENGTH];
  float msge; /* # of messages per second */
  float sema; /* # of semaphores per second */
} Msg_Sema;

typedef struct { /* report paging activity */
  char time[LENGTH];
  float vflt;  /* address translation page faults per second (valid page not in memory */
  float pflt;  /* page faults from pretection errors per second (illegal access to page) */
  float pgfil; /* vflt/s satisfied by page-in from file system */
  float rclm;  /* valid pages reclaimed for free list */
} Paging;

typedef struct { /* report unused memory pages and disk blocks */
  char time[LENGTH];
  long freemem; /* average pages available to user processes */
  long freeswp; /* disk blocks available for process swapping */
} Mem_Blk;

typedef struct { /* legend for y axis */
  char time[LENGTH];
} Y_axis;

typedef struct { /* Sa data structure for the data storage of the sarview program */
  int index;
  char sys_name[120];
  Y_axis *y_axis;

  CPU_Utilization *cpu_u;
  char cpu_legend[5][LENGTH];

  Buffer_Activity *buf;
  char buf_legend[9][LENGTH];

  Block_Device *blk_dev;
  char blk_legend[8][LENGTH];

  TTY *tty;
  char tty_legend[7][LENGTH];

  Sys_Call *sysc;
  char sysc_legend[8][LENGTH];

  Sys_Swap *syss;
  char syss_legend[6][LENGTH];

  File_Access *filea;
  char filea_legend[4][LENGTH];

  Queue_Length *q;
  char q_legend[5][LENGTH];

  Table_Status *tbs;
  char tbs_legend[8][LENGTH];

  Msg_Sema *msg;
  char msg_legend[3][LENGTH];

  Paging *p;
  char p_legend[5][LENGTH];

  Mem_Blk *mem_blk;
  char mem_blk_legend[3][LENGTH];
} Sa;

struct _choosen { /* record user's choices */
  char filename[120]; /* chosen file name */
  char items[12][80]; /* # of options selected */
  char title[80];
  char start[20];     /* the start time chosen */
  char end[20];       /* the end time chosen */
  char interval[80];  /* the amount of interval chosen */
  int num;            /* auxiliary variables */
  int positions[14];
  int pos[14];
};

typedef struct { /* parameters for drawing */
  Widget window; /* window id */
  GC gc;         /* pointer to the graphic context */
```

```
  Pixmap pixmap; /* id of a pixmap */
  Dimension width,height;  /* width and height of the window to be drawn */
} Drawing_Para;

Widget graph_box_globe; /* globel varible for setting the busy cursor */

#endif


/* getdata.c starts here */

#include "sarview.h"

/* macro for options */
#define CPU_UTILIZATION 1  /*  -u %usr   */
#define BUFFER_ACTIVITY 2  /* -b bread/s */
#define BLOCK_DEVICE    3  /* -d device  */
#define TTY_DEVICE      4  /* -y rawch/s */
#define SYSTEM_CALL     5  /* -c scalls/s */
#define SYSTEM_SWAP     6  /* -w swpin/s  */
#define FILE_ACCESS     7  /* -a iget/s   */
#define QUEUE_LENGTH    8  /* -q runq-sz  */
#define TABLE_STATUS    9  /* -v proc-sz  */
#define MES_SEMAPHORE   10 /* -m msg/s    */
#define PAGING                  11 /* -p vflt/s   */
#define MEM_PAGE_D_BLK  12 /* -r freemem  */

#define YES 1
#define NO 0

extern Sa sa;  /* global variable for storing the data generated by the sar command */
extern struct _choosen choosen; /* for storing file and option selections */

static int SIZE;

char *Short_list[] = { /* used for option listing */
  "u", "b", "d", "y",
  "c", "w", "a", "q",
  "v", "m", "p", "r"
};

/*
get_sar_data() executes sar command and fills data into
structure Sa for later graphical display. It preruns sar to get data size for dynamic
memory allocation, and then calls corresponding procedures to read in data items into Sa
structure. If read is successful, 0 is returned, otherwise, 1 is returned.
*/


int get_sar_data()
{
  int message,i,j,x,line_num;
  char cmd[120],errmsg[120],command[150];
  char str1[80],str2[80];
  char *str;
  FILE *fp;

  line_num = get_line_number(); /* get line number */
  if(line_num < 2)              /* error in read in */
     return (1);
  SIZE = line_num+2;            /* set structure size */
  clean_sa(); /* release previously allocated storage */
  sa.index = 0;
  /* set up a shell script to execute sar command */
  strcpy(cmd,"/bin/csh -fc '(sar -");
  for(i=0;i<choosen.num;i++)
    {  /* get choosen items */
    j = choosen.pos[i] - 1;
    if(j<0)
      continue;
    strcat(cmd,Short_list[j]);
    }
  if(strcmp(choosen.filename,"") !=0)
    { /* file name is specified */
    strcat(cmd," -f ");
    strcat(cmd,choosen.filename);
    }
  sprintf(command,"%s %s %s %s  > /tmp/CC.stdout) >& /tmp/CC.stderr'",
    cmd,choosen.start,choosen.end,choosen.interval);
  message = system(command); /* execute the script */
```

```
  if(message >0)
    { /* error in read in */
    if( (fp = fopen("/tmp/CC.stderr","r")) == NULL)
      { printf("\nCannot open /tmp/CC.std.err.\n"); exit(1); }
    strcpy(errmsg,"");
    while(fgets(str,80,fp) != NULL)
    strcat(errmsg,str);
    printf("\nerrmsg:\n%s\n",errmsg); /* display error msg */
    fclose(fp);
    exit(1);
    }
  if( (fp = fopen("/tmp/CC.stdout","r")) == NULL)
    { printf("\nCannot open /tmp/CC.stdout.\n"); exit(1); }

  strcpy(choosen.title,"");
  while(!feof(fp))
    { /* skip new line char */
    str = fgets(choosen.title,80,fp);
    if( (j=strlen(choosen.title)) >1) /* not new line char */
    break;
    }
  if(feof(fp))
    { printf("\nEmpty File"); return(1); }
  choosen.title[j-1] = '\0';
  strcpy(sa.sys_name,choosen.title); /* store host name */

  while(fscanf(fp, "%s %s",str1,str2) != EOF) /* until EOF */
    {
    switch (find_options(str2)) /* decide which option */
      {
      case 1: if(get_cpu_utilization(str1,str2,fp))
                 { fclose(fp); return (1); }
              break;
      case 2: if(get_buf_activity(str1,str2,fp))
                 { fclose(fp); return (1); }
              break;
      case 3: if(get_blk_activity(str1,str2,fp))
                 { fclose(fp); return (1); }
              break;
      case 4: if(get_tty_activity(str1,str2,fp))
                 { fclose(fp); return (1); }
              break;
      case 5: if(get_syscall_activity(str1,str2,fp))
                 { fclose(fp); return (1); }
              break;
      case 6: if(get_sys_swap_activity(str1,str2,fp))
                 { fclose(fp); return (1); }
              break;
      case 7: if(get_file_acc_activity(str1,str2,fp))
                 { fclose(fp); return (1); }
              break;
      case 8: if(get_queue_activity(str1,str2,fp))
                 { fclose(fp); return (1); }
              break;
      case 9: if(get_table_activity(str1,str2,fp))
                 { fclose(fp); return (1); }
              break;
      case 10: if(get_msg_activity(str1,str2,fp))
                 { fclose(fp); return (1); }
               break;
      case 11: if(get_paging_activity(str1,str2,fp))
                 { fclose(fp); return (1); }
               break;
      case 12: if(get_mem_disk_activity(str1,str2,fp))
                 { fclose(fp); return (1); }
               break;
      default: printf("\nInput Format Error\n");
               fclose(fp);
               return (1);
      }  /* switch */
    } /* while */
  fclose(fp);
  system("rm -f /tmp/CC.stderr /tmp/CC.stdout");
  return (0);  /* read in sucessfully */
} /* end of get_sar_data() */


/*
get_line_number() preruns the sar command using the selected file and gets the data size.
*/
```

```
get_line_number()
{
   int line_num = 0;
   char cmd[120],str[81],file_name[80];
   FILE *fp;

   if(strcmp(choosen.filename,"") ==0)
     strcpy(file_name,choosen.filename);
   else
     sprintf(file_name,"-f %s ",choosen.filename);
   /* set up a shell script for executing sar */
   sprintf(cmd,"/bin/csh -fc '(sar -q %s %s %s %s >
    /tmp/out.num) >& /tmp/out.err'",choosen.start,
    choosen.end,choosen.interval,file_name);

   system(cmd);
   if( (fp = fopen("/tmp/out.num","r")) == NULL)
    { printf("\nCannot open /tmp/out.num\n"); exit(1); }

   while(fgets(str,80,fp))
     line_num++;
   fclose(fp);
   system("rm /tmp/out.num /tmp/out.err"); /* delete files */
   return line_num >5 ? line_num-3: line_num-2;
} /* end of get_line_number() */


/*
find_options() gets the choosen item by matching.
*/

find_options(name)
char *name;
{
  if(strcmp(name,"%usr") == 0)
    return CPU_UTILIZATION;
  if(strcmp(name,"bread/s") ==0)
    return BUFFER_ACTIVITY;
  if(strcmp(name,"device") ==0)
    return BLOCK_DEVICE;
  if(strcmp(name,"rawch/s") ==0)
    return TTY_DEVICE;
  if(strcmp(name,"scall/s") ==0)
    return SYSTEM_CALL;
  if(strcmp(name,"swpin/s") ==0)
    return SYSTEM_SWAP;
  if(strcmp(name,"iget/s") ==0)
    return FILE_ACCESS;
  if(strcmp(name,"runq-sz") ==0)
    return QUEUE_LENGTH;
  if(strcmp(name,"proc-sz") ==0)
    return TABLE_STATUS;
  if(strcmp(name,"msg/s") ==0)
    return MES_SEMAPHORE;
  if(strcmp(name,"vflt/s") ==0)
    return PAGING;
  if(strcmp(name,"freemem") ==0)
    return MEM_PAGE_D_BLK;
  else
    return 0;
} /* end of fine_options() */


/*
get_cpu_utilization() fills data into the dynamiclly allocated CPU_Utilization structure
of Sa.
*/

get_cpu_utilization(str1,str2,fp)
char *str1, *str2; /* first and second column strings */
FILE *fp;
{
  int size,k,total=2,i=1;
  CPU_Utilization *temp;
  Y_axis *y_axis;

  if(sa.index >0)
     size = sa.index;
  else
    size = SIZE;
```

```
  temp = (CPU_Utilization*) calloc(size,sizeof (CPU_Utilization));
  if(!temp)
    { printf("\nallocation failure -- aborting\n");
      exit(1);
    }
  /* set and read in item names */
  strcpy(sa.cpu_legend[0],str1);
  strcpy(sa.cpu_legend[1],str2);
  for(k=2;k<5;k++)
    fscanf(fp,"%s",sa.cpu_legend[k]);
  if(feof(fp))
    {
    printf("\n End of File after 1st read\n");
    return (1);
    }
  while(!feof(fp))
    {
    fscanf(fp,"%s",(temp+i)->time); /* read in time */
    if(feof(fp))
      {
      printf("\n End of File after 1st read\n");
      return (1);
      }
    if(i==1) /* first read in */
      {
      if(strncmp((temp+i)->time,str1,8)==0) /* same value */
        return (1); /* empty file or format error */
      }
    if(strcmp((temp+i)->time,"Average") ==0)
      break; /* next option encountered */
    fscanf(fp,"%s %s %s %s",(temp+i)->usr,(temp+i)->sys,(temp+i)->wio,(temp+i)->idle);
    i++; total++;
    if(i>=size) /* boundary checking */
      {
      printf("\n i %d >= size %d in cpu\n",i,size);
      exit(-1);
      }
    } /* while */
  /* read in values for Average */
  fscanf(fp,"%s %s %s %s",(temp+i)->usr,(temp+i)->sys,
    (temp+i)->wio,(temp+i)->idle);
  sa.cpu_u = temp;
  if(sa.index ==0)
    sa.index = total;
  if(sa.y_axis == NULL)
    { /* set up time value array only once */
    y_axis = (Y_axis*) calloc(total,sizeof(Y_axis));
    for(i=1;i<total;i++)
    strcpy((y_axis+i)->time,(temp+i)->time);
    sa.y_axis = y_axis;
    }
  return (0);
} /* end of get_cup_utilization() */


/*
get_buf_activity() fills data into the dynamiclly allocated Buffer_Activity structure of
Sa.  The approach to reading in is similar to the one in get_cpu_utilization(), and thus
similar in-line comments are omitted.
*/

get_buf_activity(str1,str2,fp)
char *str1, *str2;
FILE *fp;
{
  int size,k,total=2,i=1;
  Buffer_Activity  *temp;
  Y_axis *y_axis;

  if(sa.index >0)
    size = sa.index;
  else
    size = SIZE;

  temp = (Buffer_Activity *) calloc(size,sizeof(Buffer_Activity));
  if(!temp)
    { printf("\nallocation failure for Buffer_Act -- aborting\n"); exit(1); }
  strcpy(sa.buf_legend[0],str1);
  strcpy(sa.buf_legend[1],str2);
  for(k=2;k<9;k++)
    fscanf(fp,"%s",sa.buf_legend[k]);
```

```
  if(!feof(fp))
    fscanf(fp,"%s",(temp+i)->time);
  else
    return (1);
  if(strncmp((temp+i)->time,sa.buf_legend[0],8) == 0)
    return (1);
  if(!feof(fp))
    fscanf(fp,"%d %d %d %d %d %d %d %d", &(temp+i)->bread,&(temp+i)->lread,
         &(temp+i)->rcache,&(temp+i)->bwrite,&(temp+i)->lwrite,
         &(temp+i)->wcache,&(temp+i)->pread,&(temp+i)->pwrite);
  else
    return (1);
  i++; total++;
  while(!feof(fp))
    {
    fscanf(fp,"%s",(temp+i)->time);
    if(strcmp((temp+i)->time,"Average") ==0)
         break;
    fscanf(fp,"%d %d %d %d %d %d %d %d", &(temp+i)->bread,&(temp+i)->lread,
         &(temp+i)->rcache,&(temp+i)->bwrite,&(temp+i)->lwrite,
         &(temp+i)->wcache,&(temp+i)->pread,&(temp+i)->pwrite);
    i++; total++;
    if(i>=size)
      { printf("\n i %d >= size %d in buf\n",i,size); exit(-1); }
    }
  fscanf(fp,"%d %d %d %d %d %d %d %d", &(temp+i)->bread,&(temp+i)->lread,
      &(temp+i)->rcache,&(temp+i)->bwrite,&(temp+i)->lwrite,
      &(temp+i)->wcache,&(temp+i)->pread,&(temp+i)->pwrite);
  sa.buf = temp;
  if(sa.index ==0)
    sa.index = total;
  if(sa.y_axis == NULL)
      {
      y_axis = (Y_axis*) calloc(total,sizeof(Y_axis));
      for(i=1;i<total;i++)
         strcpy((y_axis+i)->time,(temp+i)->time);
      sa.y_axis = y_axis;
      }
  return (0);
} /* end of get_buf_activity() */


/*
get_blk_activity() fills data into the dynamiclly allocated Block_Device structure of Sa.
The approach to reading in is similar to the one in get_cpu_utilization(), and thus
similar in-line comments are omitted.
*/

get_blk_activity(str1,str2,fp)
char *str1, *str2;
FILE *fp;
{
  int size,k,num,max;
  int total=2;
  int i=1;
  char str[80],next_char;
  Block_Device  *temp;
  Y_axis *y_axis;

  if(sa.index >0)
      size = sa.index;
  else
      size = SIZE;

  temp = (Block_Device *) calloc(size,sizeof(Block_Device));
  if(!temp)
      { printf("\nallocation failure for Block_Dev -- aborting\n"); exit(1); }
  strcpy(sa.blk_legend[0],str1);
  strcpy(sa.blk_legend[1],str2);
  for(k=2;k<8;k++)
      fscanf(fp,"%s",sa.blk_legend[k]);
  num= (-1); max = (-1);
  fscanf(fp,"%s",(temp+i)->time);
  if(feof(fp))
      {
      printf("\n End of File after 1st read\n");
      return (1);
      }
  if(strncmp((temp+i)->time,sa.blk_legend[0],8) == 0)
      return (1);
  k=0;
```

```
  while(!feof(fp))
      {
      fscanf(fp,"%s",str);
      if(str[2] == ':' || *str=='A')
         { /* next device */ (temp+i)->dev_no = k;
         if(k>max) max = k; /* get max number */
         k=0;
         i++; total++;
         if(strcmp(str,"Average") ==0)
           num = max;
         strcpy((temp+i)->time,str);
         fscanf(fp,"%s %d %f %d %d %f %f",(temp+i)->data[k].devi ce,
           &(temp+i)->data[k].busy,&(temp+i)->data[k].avque,
           &(temp+i)->data[k].r_w,&(temp+i)->data[k].blks,
           &(temp+i)->data[k].avwait,&(temp+i)->data[k].avserv);
         k++;
         }
      else
         {
         strcpy((temp+i)->data[k].device,str);
         fscanf(fp,"%d %f %d %d %f %f",&(temp+i)->data[k].busy,&(temp+i)->data[k].avque,
           &(temp+i)->data[k].r_w,&(temp+i)->data[k].blks,
           &(temp+i)->data[k].avwait,&(temp+i)->data[k].avserv);
         k++;
         if(k==num)
           {
           next_char = getc(fp);
           while( !isdigit(next_char) && next_char != EOF )
                {
                if(isspace(next_char)) /* skip space */
                   ;
                else
                  {
                  ungetc(next_char,fp);
                  fscanf(fp,"%s %d %f %d %d %f %f", (temp+i)->data[k].device,
                    &(temp+i)->data[k].busy, &(temp+i)->data[k].avque,
                    &(temp+i)->data[k].r_w, &(temp+i)->data[k].blks,
                    &(temp+i)->data[k].avwait, &(temp+i)->data[k].avserv);
                  k++;
                  }
                next_char = getc(fp);
                }
           ungetc(next_char,fp);
           (temp+i)->dev_no = k;
           break;
           }
         }
    } /* while */
  sa.blk_dev = temp;
  if(sa.index ==0)
    sa.index = total;
  if(sa.y_axis == NULL)
      { y_axis = (Y_axis*) calloc(total,sizeof(Y_axis));
      for(i=1;i<total;i++)
         strcpy((y_axis+i)->time,(temp+i)->time);
      sa.y_axis = y_axis;
      }
  return (0);
} /* end of get_blk_activity() */


/*
get_tty_activity() fills data into the dynamiclly allocated TTY structure of Sa. The
approach to reading in is similar to the one in get_cpu_utilization(), and thus similar
in-line comments are omitted.
*/

get_tty_activity(str1,str2,fp)
char *str1, *str2;FILE *fp;
{
  int size,k,total=2,i=1;
  TTY  *temp;
  Y_axis *y_axis;

  if(sa.index >0)
      size = sa.index;
  else
      size = SIZE;

  temp = (TTY *) calloc(size,sizeof(TTY));
  if(!temp)
```

```
    { printf("\nallocation failure for TTY -- aborting\n"); exit(1); }
  strcpy(sa.tty_legend[0],str1);
  strcpy(sa.tty_legend[1],str2);
  for(k=2;k<7;k++)
    fscanf(fp,"%s",sa.tty_legend[k]);
  fscanf(fp,"%s",(temp+i)->time);
  if(feof(fp))
   {
   printf("\n End of File after 1st read\n");
   return (1);
   }
  if(strncmp((temp+i)->time,sa.tty_legend[0],8) == 0)
    return (1);
  fscanf(fp,"%d %d %d %d %d %d ",&(temp+i)->rawch,
    &(temp+i)->canch,&(temp+i)->outch,&(temp+i)->rcvin,
    &(temp+i)->xmtin,&(temp+i)->mdmin);
  i++; total++;
  while(!feof(fp))
    {
    fscanf(fp,"%s",(temp+i)->time);
    if(strcmp((temp+i)->time,"Average") ==0)
        break;
    fscanf(fp,"%d %d %d %d %d %d ", &(temp+i)->rawch,&(temp+i)->canch,
        &(temp+i)->outch,&(temp+i)->rcvin,
        &(temp+i)->xmtin,&(temp+i)->mdmin);
    i++; total++;
    if(i>=size)
        { printf("\n i %d >= size %d in tty \n",i,size);exit(-1);}
    }
  fscanf(fp,"%d %d %d %d %d %d ",&(temp+i)->rawch,
      &(temp+i)->canch,&(temp+i)->outch,&(temp+i)->rcvin,
      &(temp+i)->xmtin,&(temp+i)->mdmin);
  sa.tty = temp;
  if(sa.index ==0)
      sa.index = total;
  if(sa.y_axis == NULL)
      {
      y_axis = (Y_axis*) calloc(total,sizeof(Y_axis));
      for(i=1;i<total;i++)
         strcpy((y_axis+i)->time,(temp+i)->time);
      sa.y_axis = y_axis;
      }
  return (0);
}  /* end of get_tty_activity() */


/*
get_syscall_activity() fills data into the dynamiclly allocated Sys_Call structure of Sa.
The approach to reading in is similar to the one in get_cpu_utilization(), and thus
similar in-line comments are omitted.
*/

get_syscall_activity(str1,str2,fp)
char *str1, *str2;
FILE *fp;
{
  int size,k,total=2,i=1;
  Sys_Call  *temp;
  Y_axis *y_axis;

  if(sa.index >0)
    size = sa.index;
  else
    size = SIZE;

  temp = (Sys_Call *) calloc(size,sizeof(Sys_Call));
  if(!temp)
    { printf("\nallocation failure for Sys_Call --aborting\n"); exit(1); }
  strcpy(sa.sysc_legend[0],str1);
  strcpy(sa.sysc_legend[1],str2);
  for(k=2;k<8;k++)
    fscanf(fp,"%s",sa.sysc_legend[k]);
  fscanf(fp,"%s",(temp+i)->time);
  if(feof(fp))
   {
   printf("\n End of File after 1st read\n");
   return (1);
   }
  if(strncmp((temp+i)->time,sa.sysc_legend[0],8) == 0)
    return (1);
  fscanf(fp,"%d %d %d %f %f %ld %ld", &(temp+i)->scall,&(temp+i)->sread,
```

```
        &(temp+i)->swrit,&(temp+i)->fork,&(temp+i)->exec,
        &(temp+i)->rchar,&(temp+i)->wchar);
    i++; total++;
    while(!feof(fp))
        {
        fscanf(fp,"%s",(temp+i)->time); if(strcmp((temp+i)->time,"Average") ==0)
           break;
        fscanf(fp,"%d %d %d %f %f %ld %ld", &(temp+i)->scall,&(temp+i)->sread,
           &(temp+i)->swrit,&(temp+i)->fork,&(temp+i)->exec,
           &(temp+i)->rchar,&(temp+i)->wchar);
        i++; total++;
        if(i>=size)
           { printf("\n i %d >= size %d in sysc\n",i,size);exit(-1);}
        }
    fscanf(fp,"%d %d %d %f %f %ld %ld", &(temp+i)->scall,&(temp+i)->sread,
        &(temp+i)->swrit,&(temp+i)->fork,&(temp+i)->exec,
        &(temp+i)->rchar,&(temp+i)->wchar);
    sa.sysc = temp;
    if(sa.index ==0)
        sa.index = total;
    if(sa.y_axis == NULL)
        {
        y_axis = (Y_axis*) calloc(total,sizeof(Y_axis));
        for(i=1;i<total;i++)
           strcpy((y_axis+i)->time,(temp+i)->time);
        sa.y_axis = y_axis;
        }
    return (0);
}  /* end of get_syscall_activity() */


/*
get_sys_swap_activity() fills data into the dynamiclly allocated Sys_Swap structure of Sa.
The approach to reading in is similar to the one in get_cpu_utilization(), and thus
similar in-line comments are omitted.
*/

get_sys_swap_activity(str1,str2,fp)
char *str1, *str2;
FILE *fp;
{
    int size,k,total=2,i =1;
    Sys_Swap  *temp;
    Y_axis *y_axis;

    if(sa.index >0)
      size = sa.index;
    else
      size = SIZE;

    temp = (Sys_Swap *) calloc(size,sizeof(Sys_Swap));
    if(!temp)
      { printf("\nallocation failure for Sys_Swap -- aborting\n"); exit(1); }
    strcpy(sa.syss_legend[0],str1);
    strcpy(sa.syss_legend[1],str2);
    for(k=2;k<6;k++)
      fscanf(fp,"%s",sa.syss_legend[k]);
    fscanf(fp,"%s",(temp+i)->time);
    if(feof(fp))
     return (1);

    if(strncmp((temp+i)->time,sa.syss_legend[0],8) == 0) return (1);
    fscanf(fp,"%f %f %f %f %d",&(temp+i)->swpin,
          &(temp+i)->bswin,&(temp+i)->swpot,&(temp+i)->bswot,
          &(temp+i)->pswch);
    i++; total++;
    while(!feof(fp))
        {
        fscanf(fp,"%s",(temp+i)->time);
        if(strcmp((temp+i)->time,"Average") ==0)
           break;
        fscanf(fp,"%f %f %f %f %d",&(temp+i)->swpin,
           &(temp+i)->bswin,&(temp+i)->swpot,
           &(temp+i)->bswot,&(temp+i)->pswch);
        i++; total++;
        if(i>=size)
           { printf("\n i %d >= size %d in syss\n",i,size);
             exit(-1);}
        }
    fscanf(fp,"%f %f %f %f %d",&(temp+i)->swpin,
        &(temp+i)->bswin,&(temp+i)->swpot,
```

```
        &(temp+i)->bswot,&(temp+i)->pswch);
    sa.syss = temp;
    if(sa.index ==0)
        sa.index = total;
    if(sa.y_axis == NULL)
        {
        y_axis = (Y_axis*) calloc(total,sizeof(Y_axis));
        for(i=1;i<total;i++)
           strcpy((y_axis+i)->time,(temp+i)->time);
        sa.y_axis = y_axis;
        }
    return (0);
} /* end of get_sys_swap_activity() */


/*
get_file_acc_activity() fills data into the dynamiclly allocated File_Access structure of
Sa. The approach to reading in is similar to the one in get_cpu_utilization(), and thus
similar in-line comments are omitted.
*/

get_file_acc_activity(str1,str2,fp)
char *str1, *str2;
FILE *fp;
{
    int size,k,total=2,i =1;
    File_Access  *temp;
    Y_axis *y_axis;

    if(sa.index >0)      size = sa.index;
    else
        size = SIZE;

    temp = (File_Access *) calloc(size,sizeof(File_Access));
    if(!temp)
        { printf("\nallocation failure for File_Access -- aborting\n"); exit(1); }
    strcpy(sa.filea_legend[0],str1);
    strcpy(sa.filea_legend[1],str2);
    for(k=2;k<4;k++)
      fscanf(fp,"%s",sa.filea_legend[k]);
    fscanf(fp,"%s",(temp+i)->time);
    if(feof(fp))
        return (1);
    if(strncmp((temp+i)->time,sa.filea_legend[0],8) == 0)
        return (1);
    fscanf(fp,"%d %d %d",&(temp+i)->iget,&(temp+i)->namei,&(temp+i)->dirbk);
    i++; total++;

    while(!feof(fp))
        {
        fscanf(fp,"%s",(temp+i)->time);
        if(strcmp((temp+i)->time,"Average") ==0)
           break;
        fscanf(fp,"%d %d %d",&(temp+i)->iget,&(temp+i)->namei,&(temp+i)->dirbk);
        i++; total++;
        if(i>=size)
           { printf("\n i %d >= size %d in filea\n",i,size); exit(-1);}
        }
    fscanf(fp,"%d %d %d",&(temp+i)->iget,&(temp+i)->namei,&(temp+i)->dirbk);
    sa.filea = temp;
    if(sa.index ==0)
        sa.index = total;
    if(sa.y_axis == NULL)
        {
        y_axis = (Y_axis*) calloc(total,sizeof(Y_axis));
        for(i=1;i<total;i++)
           strcpy((y_axis+i)->time,(temp+i)->time);
        sa.y_axis = y_axis;
        }
    return (0);
}  /* end of get_file_acc_activity() */


/*
get_queue_activity() fills data into the dynamiclly allocated Queue_Length structure of
Sa. The approach to reading in is similar to the one in get_cpu_utilization(), and thus
similar in-line comments are omitted.
*/

get_queue_activity(str1,str2,fp)
char *str1, *str2;
```

```
FILE *fp;
{
  int size,k,total=2,i=1,indi=0,flag=1;
  char str[80];
  Queue_Length  *temp;
  Y_axis *y_axis;

  if(sa.index >0)
    size = sa.index;
  else
    size = SIZE;

  temp = (Queue_Length *) calloc(size,sizeof(Queue_Length));
   if(!temp)
    { printf("\nallocation failure for Queue_Length -- aborting\n"); exit(1); }
  strcpy(sa.q_legend[0],str1);
  strcpy(sa.q_legend[1],str2);
  for(k=2;k<5;k++)
    fscanf(fp,"%s",sa.q_legend[k]);
  if(feof(fp))
   return (1);

  while(!feof(fp))
    {
    if(flag==1)
         {
         fscanf(fp,"%s",(temp+i)->time);
         if(strcmp((temp+i)->time,"Average")==0)
              break;
         }
    else
         strcpy((temp+i)->time,str);
    if(i==1)
         {
         if(feof(fp))
             return (1);
         if(strncmp((temp+i)->time,sa.q_legend[0],8) ==0)
             return (1);
         }
    fscanf(fp,"%f %d",&(temp+i)->runq_sz,&(temp+i)->runocc);
    fscanf(fp,"%s",str);
    if(strcmp(str,"Average")==0)
         {
         i++; total++;
         strcpy((temp+i)->time,str);
         break;
         }
    if(strstr(str,":")!=0)
         { /* no swpq_sz */     (temp+i)->swpq_sz= (-1);
         flag=0;
         }
    else
         { /* store swpq_sz */
         (temp+i)->swpq_sz=atof(str);
         fscanf(fp,"%d",&(temp+i)->swpocc);
         flag=1;
         if(indi==0)
             {
             if((temp+i)->swpq_sz > 0.0)
             indi=1;
             }
         }
    i++; total++;
    if(i>=size)
        {printf("\n i %d >= size %d in q\n",i,size); exit(-1);}
      }
  fscanf(fp,"%f %d",&(temp+i)->runq_sz,&(temp+i)->runocc);
  if(indi==1) /* at least one swpq_sz */
      fscanf(fp,"%f %d",&(temp+i)->swpq_sz,&(temp+i)->swpocc);
  else
      (temp+i)->swpq_sz = (-1);
  sa.q = temp;
  if(sa.index ==0)
      sa.index = total;
  if(sa.y_axis == NULL)
      {
      y_axis = (Y_axis*) calloc(total,sizeof(Y_axis));
      for(i=1;i<total;i++)
         strcpy((y_axis+i)->time,(temp+i)->time);
      sa.y_axis = y_axis;
      }
```

```
  return (0);
}  /* end of get_queue_activity() */


/*
get_table_activity() fills data into the dynamiclly allocated Table_Status structure of
Sa. The approach to reading in is similar to the one in get_cpu_utilization(), and thus
similar in-line comments are omitted.
*/

get_table_activity(str1,str2,fp)
char *str1, *str2;
FILE *fp;
{
  int size,k,total=2,i=1;
  Table_Status  *temp;
  Y_axis *y_axis;

  if(sa.index >0)
    size = sa.index;
  else
    size = SIZE;

  temp = (Table_Status *) calloc(size,sizeof(Table_Status));
  if(!temp)
    { printf("\nallocation failure for Table_Status -- aborting\n"); exit(1); }
  strcpy(sa.tbs_legend[0],str1);
  strcpy(sa.tbs_legend[1],str2);
  for(k=2;k<8;k++)
    fscanf(fp,"%s",sa.tbs_legend[k]);
  fscanf(fp,"%s",(temp+i)->time);
  if(feof(fp))
    return (1);

  if(strncmp((temp+i)->time,sa.tbs_legend[0],8) == 0)
    return (1);
  fscanf(fp,"%d/%d %d %d/%d %d %d/%d %d %d/%d",
    &(temp+i)->proc_sz,&(temp+i)->proc_tsz,
    &(temp+i)->proc_ov,&(temp+i)->inod_sz,
    &(temp+i)->inod_tsz,&(temp+i)->inod_ov,
    &(temp+i)->file_sz,&(temp+i)->file_tsz,
    &(temp+i)->file_ov,&(temp+i)->lock_sz,
    &(temp+i)->lock_tsz);
  i++;total++;

  while(!feof(fp))
    {
    fscanf(fp,"%s",(temp+i)->time);
    if(strcmp((temp+i)->time,"Average") ==0)
         break;
    if(feof(fp))
         break;
    fscanf(fp,"%d/%d %d %d/%d %d %d/%d %d %d/%d", &(temp+i)->proc_sz,&(temp+i)->proc_tsz,
         &(temp+i)->proc_ov,&(temp+i)->inod_sz,
         &(temp+i)->inod_tsz,&(temp+i)->inod_ov,
         &(temp+i)->file_sz,&(temp+i)->file_tsz,
         &(temp+i)->file_ov,&(temp+i)->lock_sz,
         &(temp+i)->lock_tsz);
    if(feof(fp))
         break;
    i++; total++;
    if(i>=size)
      { printf("\n i %d >= size %d in tbs\n",i,size); exit(-1);}
    }
  sa.tbs = temp;
  if(sa.index ==0)
    sa.index = total;
  if(sa.y_axis == NULL)
    {
    y_axis = (Y_axis*) calloc(total,sizeof(Y_axis));
    for(i=1;i<total;i++)
       strcpy((y_axis+i)->time,(temp+i)->time);
    sa.y_axis = y_axis;
    }
  return (0);
}  /* end of get_table_activity() */


/*
get_msg_activity() fills data into the dynamiclly allocated Msg_Sema structure of Sa. The
approach to reading in is similar to the one in get_cpu_utilization(), and thus similar
```

```
in-line comments are omitted.
*/

get_msg_activity(str1,str2,fp)
char *str1, *str2;
FILE *fp;
{
  int size,k,total=2,i=1;
  Msg_Sema  *temp;
  Y_axis *y_axis;

  if(sa.index >0)
    size = sa.index;
  else
    size = SIZE;

  temp = (Msg_Sema *) calloc(size,sizeof(Msg_Sema));
  if(!temp)
    { printf("\nallocation failure for Msg_Sema-- aborting\n"); exit(1); }
  strcpy(sa.msg_legend[0],str1);
  strcpy(sa.msg_legend[1],str2);
  for(k=2;k<3;k++) fscanf(fp,"%s",sa.msg_legend[k]);
  fscanf(fp,"%s",(temp+i)->time);
  if(feof(fp))
      return (1);

  if(strncmp((temp+i)->time,sa.msg_legend[0],8) == 0)
      return (1);
  fscanf(fp,"%f %f ",&(temp+i)->msge,&(temp+i)->sema);
  i++; total++;

  while(!feof(fp))
      {
      fscanf(fp,"%s",(temp+i)->time);
      if(strcmp((temp+i)->time,"Average") ==0)
        break;
      fscanf(fp,"%f %f ",&(temp+i)->msge,&(temp+i)->sema);
      i++; total++;
      if(i>=size)
        { printf("\n i %d >= size %d in msg\n",i,size); exit(-1); }
      }
  fscanf(fp,"%f %f",&(temp+i)->msge,&(temp+i)->sema);
  sa.msg = temp;
  if(sa.index ==0)
      sa.index = total;
  if(sa.y_axis == NULL)
      {
      y_axis = (Y_axis*) calloc(total,sizeof(Y_axis));
      for(i=1;i<total;i++)
        strcpy((y_axis+i)->time,(temp+i)->time);
      sa.y_axis = y_axis;
      }
  return (0);
}  /* end of get_msg_activity() */


/*
get_paging_activity() fills data into the dynamiclly allocated Paging structure of Sa. The
approach to reading in is similar to the one in get_cpu_utilization(), and thus similar
in-line comments are omitted.
*/

get_paging_activity(str1,str2,fp)
char *str1, *str2;
FILE *fp;
{
  int size,k,total=2,i=1;
  Paging  *temp;
  Y_axis *y_axis;

  if(sa.index >0)
    size = sa.index;
  else
    size = SIZE;

  temp = (Paging *) calloc(size,sizeof(Paging));
  if(!temp)
    { printf("\nallocation failure for Paging -- aborting\n"); exit(1); }
  strcpy(sa.p_legend[0],str1);
    strcpy(sa.p_legend[1],str2);
  for(k=2;k<5;k++)
```

```
    fscanf(fp,"%s",sa.p_legend[k]);
  fscanf(fp,"%s",(temp+i)->time);
  if(feof(fp))
    return (1);

  if(strncmp((temp+i)->time,sa.p_legend[0],8) == 0)
    return (1);
  fscanf(fp,"%f %f %f %f ",&(temp+i)->vflt,
        &(temp+i)->pflt,&(temp+i)->pgfil,&(temp+i)->rclm);
  i++; total++;

  while(!feof(fp))
    {
    fscanf(fp,"%s",(temp+i)->time);
    if(strcmp((temp+i)->time,"Average") ==0)
      break;
    fscanf(fp,"%f %f %f %f ",&(temp+i)->vflt,
        &(temp+i)->pflt,&(temp+i)->pgfil,&(temp+i)->rclm);
    i++; total++;
    if(i>=size)
      { printf("\n i %d >= size %d in paging\n",i,size); exit(-1);}
    }
  fscanf(fp,"%f %f %f %f ",&(temp+i)->vflt,
    &(temp+i)->pflt,&(temp+i)->pgfil,&(temp+i)->rclm);
  sa.p = temp;
  if(sa.index ==0)
    sa.index = total;
  if(sa.y_axis == NULL)
    {
    y_axis = (Y_axis*) calloc(total,sizeof(Y_axis));
    for(i=1;i<total;i++)
    strcpy((y_axis+i)->time,(temp+i)->time);
    sa.y_axis = y_axis;
    }
  return (0);
} /* end of get_paging_activity() */


/*
get_mem_disk_activity() fills data into the dynamiclly allocated Mem_Blk structure of Sa.
The approach to reading in is similar to the one in get_cpu_utilization(), and thus
similar in-line comments are omitted.
*/

get_mem_disk_activity(str1,str2,fp)
char *str1, *str2;
FILE *fp;
{
  int size,k,total=2,i=1;
  Mem_Blk  *temp;
  Y_axis *y_axis;

  if(sa.index >0)
    size = sa.index;
  else
    size = SIZE;

  temp = (Mem_Blk *) calloc(size,sizeof(Mem_Blk));
  if(!temp)
    { printf("\nallocation failure for Mem_Blk-- aborting\n"); exit(1); }
  strcpy(sa.mem_blk_legend[0],str1);
  strcpy(sa.mem_blk_legend[1],str2);
  for(k=2;k<3;k++)
    fscanf(fp,"%s",sa.mem_blk_legend[k]);
  fscanf(fp,"%s",(temp+i)->time);
  if(feof(fp))
    return (1);

  if(strncmp((temp+i)->time,sa.mem_blk_legend[0],8) == 0)
    return (1);
  fscanf(fp,"%ld %ld ",&(temp+i)->freemem,&(temp+i)->freeswp);
  i++; total++;

  while(!feof(fp))
    {
    fscanf(fp,"%s",(temp+i)->time);
    if(strcmp((temp+i)->time,"Average") ==0)
    break;
    fscanf(fp,"%ld %ld ",&(temp+i)->freemem,&(temp+i)->freeswp);
    i++; total++;
    if(i>=size)
```

```
      { printf("\n i %d >= size %d in mem_blk\n",i,size); exit(-1);}
    }
  fscanf(fp,"%ld %ld ",&(temp+i)->freemem,&(temp+i)->freeswp);
  sa.mem_blk = temp;
  if(sa.index ==0)
    sa.index = total;
  if(sa.y_axis == NULL)
    {
    y_axis = (Y_axis*) calloc(total,sizeof(Y_axis));
    for(i=1;i<total;i++)
    strcpy((y_axis+i)->time,(temp+i)->time);
    sa.y_axis = y_axis;
    }
  return (0);
} /* end of get_mem_disk_activity() */


/**** motif.h ****/
/* X and Motif include files for menu.c and graph.c */

#ifndef H_MOTIF
#define H_MOTIF

/* include X library files as required */
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <X11/StringDefs.h>
#include <X11/Xutil.h>
#include <X11/cursorfont.h>
#include <Xm/BulletinB.h>
#include <Xm/CascadeB.h>
#include <Xm/CascadeBG.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/DrawnB.h>
#include <Xm/FileSB.h>
#include <Xm/Form.h>
#include <Xm/Frame.h>
#include <Xm/Label.h>
#include <Xm/LabelG.h>
#include <Xm/List.h>
#include <Xm/MainW.h>
#include <Xm/MessageB.h>
#include <Xm/PanedW.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>
#include <Xm/RowColumn.h>
#include <Xm/ScrolledW.h>
#include <Xm/SelectioB.h>
#include <Xm/SeparatoG.h>
#include <Xm/Separator.h>
#include <Xm/Text.h>
#include <Xm/ToggleB.h>
#include <Xm/ToggleBG.h>

#endif


/* menu.h starts here */
#ifndef H_MENU
#define H_MENU

#define NUM_ARGS 20 /* max. size of argument list */

/******* Main Program ********/

#define PROGRAM_CLASS "MGUI"

#define YES 1
#define NO 0
#define MAIN_TITLE "System Activity Graphical Displayer"
#define charset  XmSTRING_DEFAULT_CHARSET
#define BUF_SIZE 300

char *List[] = {/* list of options for both Options and Grphics selection windows */
  "CPU Utlization",
  "Buffer Activity",
  "Block Device Activity",
  "TTY Device Activity",
```

```
  "System Calls",
  "System Swapping & Switching",
  "File Access System Routine Usage",
  "Average Queue Length",
  "Status of Process, Inode, File Tables",
  "Message & Semaphore Activities",
  "Paging Activities",
  "Unused Memory Pages & Disk Blocks"
};

char *prompts[] = { /* menu entries for start time, end time, and amount of interval */
  "Start  (hh:mm):",
  "End    (hh:mm):",
  "Interval (ss):",
};

String description_text[] = {
 "This is a graphical display of system performance based on data",
 "obtained by the sar command. For the sar data details, please refer to the man page"
  "for sar manual. The programmer, Haibo Du, welcomes any suggestion and comment."
  "Email: dhaibo@a.cs.okstate.edu",
   NULL
};


struct _choosen choosen; /* for file and options selection */


Sa sa; /* for storing performance data */
Drawing_Para draw_para[12]; /* store information about the drawing window */


/* main program */

XmString MakeXmString();
void show_fselbox();
void show_opselbox();
void show_graphselbox();
void graph_callback();
void popdown();
void unset_state();
void quit();
void popdownFile();
void get_choices();
void get_strings();
void my_proc();
void gracmd();
void popdownbox();
void fselbox_default();
void fselbox_cancel();
void init_draw_para();
char* MakeCharString();
void create_description_w();
Widget create_menu();
Widget create_fselbox();
Widget create_opselbox();
Widget create_graphselbox();
Widget create_error_box();
int get_sar_data();
Cursor create_working_cursor();
void display_working_cursor();
void undisplay_working_cursor();


XmString XmList[10]; /* stores Xm strings */

Widget gra_cmd[12]; /* widget ids for buttons in the Graphics window */
Widget toggleset;
Widget list_w; /* id to text list window */
Cursor working_cursor; /* id to busy cursor */
static Widget error_box=NULL; /* default set for pointer to error box */

#endif



/* menu.c starts here */
#include "sarview.h"
#include "motif.h"
#include "menu.h"
```

```
/*
main() displays graphically the Sequent S81 system performance by using sar. This is done
through graphical interaction with a user by calling corresponding functions to create
graphical objects and processing of a user's query. */

main(argc,argv)
unsigned int argc;
char **argv;
{
  Display *display;
  XtAppContext context;
  Arg args[NUM_ARGS];
  Widget toplevel,menu,form,
  int n,i;

  /* set default dir */
  XmString default_dir = MakeXmString("/usr/adm/sa/");

  /* initialize toolkit */
  XtToolkitInitialize();
  context = XtCreateApplicationContext();
  display = XtOpenDisplay (
                  context, /* application context  */
                  NULL,    /* use DISPLAY from environment */
                  NULL,    /* use last of argv[0] as name */
                  PROGRAM_CLASS,
                  NULL,    /* no additional command line options */
                  NULL,    /* ditto */
                  &argc,
                  argv); /* use and delete standard options */

  if(display == NULL) /* check for display connection */
    { printf("cannot open display"); exit(-1); }

  printf("\nLoading ...\n");

  /* initialize application parameters */
  init_sa();
  /* create application shell */
  n=0; /* set menu window title */
  XtSetArg(args[n],XmNtitle,MAIN_TITLE); n++;
  toplevel = XtAppCreateShell(
        NULL, /* use same program name */
        PROGRAM_CLASS,
        applicationShellWidgetClass,
        display,
        args,n); n=0; /* set menu window size */
  XtSetArg(args[n],XmNwidth,518); n++;
  XtSetArg(args[n],XmNheight,119);n++;
  form = XtCreateManagedWidget("form",xmFormWidgetClass,
         toplevel,args,n);

  /* create graphical objects within the form widget */
  menu = create_menu(form); /* create menu window */
  /* create description window */
  create_description_w(form,menu);
  working_cursor = create_working_cursor(toplevel);

  /* start loop */
  XtRealizeWidget(toplevel);
  XtAppMainLoop(context);
} /* end of main */


/*
create_menu() creates a main menu window which consists of frame, rowcolumn, radiobox and
toggle button widgets. The menu has four entries: file, options, graphics, and quit. It
returns a pointer to the menu widget.
*/

Widget create_menu(parent)
Widget parent; /* form widget */
{
  Widget frame,rowcol,commands,button[4];
  Widget fselbox, opselbox, graphselbox;
  Arg args[NUM_ARGS];
  int n;
```

```
  XmString str;

  n=0; /* place the menu frame left side of the form */
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNshadowType,XmSHADOW_IN); n++;
  frame = XtCreateWidget("frame",xmFrameWidgetClass,parent,args,n);

  /* create rowcolumn widget as child of frame widget */
  /* to ease the layout of the entries                */
  rowcol = XtCreateWidget("rowcol",xmRowColumnWidgetClass,frame,NULL,0);
  n=0; /* create radiobox for one choice at a time */
  XtSetArg(args[n],XmNentryClass,xmToggleButtonWidgetClass); n++;
  XtSetArg(args[n],XmNradioBehavior,True);n++;
  XtSetArg(args[n],XmNradioAlwaysOne,True);n++;
  commands = XmCreateRadioBox(rowcol,"commands",args,n);

  /* create a file button */
  button[0] = XtCreateManagedWidget("File",xmToggleButtonWidgetClass,commands,args,0);
  /* create file selection box related to the file button*/
  fselbox = create_fselbox(button[0]);
  /* register callback related to the file button widget */
  XtAddCallback(button[0],XmNvalueChangedCallback,
   show_fselbox,fselbox);
  /* create an options button */
  button[1] = XtCreateManagedWidget("Options",
   xmToggleButtonWidgetClass,commands,args,0);
  /* create an options selection box */
  opselbox = create_opselbox(button[1]);
   /* register callback related to the options button widget */
  XtAddCallback(button[1],XmNvalueChangedCallback,
   show_opselbox,opselbox);
  toggleset = button[1];
  /* create a graphics button */
  button[2] = XtCreateManagedWidget("Graphics",
   xmToggleButtonWidgetClass,commands,NULL,0);
  /* create a graphics selection box */
  graphselbox = create_graphselbox(button[2]);
   /* register callback related to the graphics button widget */
  XtAddCallback(button[2],XmNvalueChangedCallback,
   show_graphselbox,graphselbox);
  graph_box_globe = graphselbox; /* assign to global var */
  /* create a quit button */
  button[3] = XtCreateManagedWidget("Quit",xmToggleButtonWidgetClass,commands,args,0);
  /* register callback related to the quit button widget */
  XtAddCallback(button[3],XmNvalueChangedCallback,
   quit,XtParent(parent));

  /* manage widgets */
  XtManageChild(commands);
  XtManageChild(rowcol);
  XtManageChild(frame);

  return (frame);
} /* end of create_menu() */


/*
show_graphselbox() pops up the graphics selection box.
*/

void show_graphselbox(w,dia_shell,call_data)
Widget w;
Widget dia_shell; /* widget of graphics box */
XmAnyCallbackStruct *call_data;
{
  Arg args[NUM_ARGS];
  int n,i;

  for(i=0; i<XtNumber(List); i++)
    { /* set insensitive all the buttons in the box */
    n=0;
    XtSetArg(args[n],XmNsensitive,False); n++;
    XtSetValues(gra_cmd[i],args,n);
    }
```

```
  if(choosen.num >0) /* check number of items choosen */
      {
       for(i=0;i<choosen.num;i++)
         { /* set sensitive those buttons choosen */
         n=0;
         XtSetArg(args[n],XmNsensitive,True); n++;
         XtSetValues(gra_cmd[choosen.pos[i]-1],args,n);
         }
      }
  XtManageChild(dia_shell);
}  /* end of show_graphselbox() */


/*
create_graphselbox() creates a graphics selection box which has widgets of
BulletinBoardDialog, pane, radiobox, toggle_button. It returns a pointer to the graphics
selection box widget.
*/

Widget create_graphselbox(parent)
Widget parent; /* button[2]  -- graphics button */
{
  Widget dia_shell,pane,gra_box,quit_box,quit_pb;
  Arg args[NUM_ARGS];
  int n,i;

  n=0; /* set dialog style, title */
  XtSetArg(args[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL);n++;
  XtSetArg(args[n],XmNdialogTitle,MakeXmString("Graphics Selection"));n++;
  XtSetArg(args[n],XmNmarginWidth,1);n++;
  dia_shell = XmCreateBulletinBoardDialog(parent,"bulletin",args,n);
  n=0; /* set sash size of the pane window */
  XtSetArg(args[n],XmNsashWidth,1); n++;
  XtSetArg(args[n],XmNsashHeight,1); n++;
  pane = XtCreateWidget("pane",xmPanedWindowWidgetClass,dia_shell,args,n);
  n=0; /* set radio box behavior */
  XtSetArg(args[n],XmNentryClass,xmToggleButtonWidgetClass);n++;
  XtSetArg(args[n],XmNradioBehavior,False);n++;
  XtSetArg(args[n],XmNradioAlwaysOne,False);n++;
  gra_box = XmCreateRadioBox(pane,"gra_box",args,n);

  for(i = 0; i<XtNumber(List); i++)
    { n=0; /* create tobble button as insensitive */ XtSetArg(args[n],XmNsensitive,False);
      n++;
      gra_cmd[i] = XtCreateManagedWidget(List[i],
         xmToggleButtonWidgetClass,gra_box,args,n);
      /* register callback related to item selection widget */
      XtAddCallback(gra_cmd[i],XmNvalueChangedCallback,display_working_cursor,NULL);
      XtAddCallback(gra_cmd[i],XmNvalueChangedCallback,graph_callback,i);
    }
  /* create quit box */
  n=0;
  XtSetArg(args[n],XmNentryClass,xmToggleButtonWidgetClass);n++;
  XtSetArg(args[n],XmNradioBehavior,False);n++;
  XtSetArg(args[n],XmNradioAlwaysOne,False);n++;
  quit_box = XmCreateRadioBox(pane,"quit_box",args,n);
  /* create quit button */
  quit_pb = XtCreateManagedWidget("Quit",xmToggleButtonWidgetClass,quit_box,args,0);
  XtAddCallback(quit_pb,XmNvalueChangedCallback,popdown,dia_shell);
  XtAddCallback(dia_shell,XmNunmapCallback,unset_state,parent);
  XtManageChild(gra_box);
  XtManageChild(quit_box);

  /* Fix the action area pane to its current height */
  {
    Dimension h;
    n=0;
    XtSetArg(args[n],XmNheight,&h); n++;
    XtGetValues(quit_pb,args,n);

    n=0;
    XtSetArg(args[n],XmNpaneMinimum,h+10); n++;
    XtSetArg(args[n],XmNpaneMaximum,h+10); n++;
    XtSetValues(quit_box,args,n);
  }
  XtManageChild(pane);
  return (dia_shell);
```

```
} end of create_graphselbox() */


/*
create_description_w() creates description window which has a label widget and text
widget.
*/


void create_description_w(parent,left_w)
Widget parent,left_w; /* form and frame widgets */
{
  Widget label,text_w;
  Arg args[NUM_ARGS];  int n,i;
  XmString str;
  char buf[BUF_SIZE],*p,str1[81];
  FILE *fp;

  /* description label */
  str = MakeXmString("Description");
  n=0; /* place label top and right to menu */
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNleftWidget,left_w); n++;
  XtSetArg(args[n],XmNlabelString,str);n++;
  label = XtCreateManagedWidget("label",xmLabelWidgetClass,parent,args,n);
  if( (fp = fopen("description.text","r")) == NULL)
    { /* check for file open */
    printf("\nCannot open description.text\n");
    for(p=buf,i=0; description_text[i]; i++)
    {
    p += strlen(strcpy(p,description_text[i]));
    if( !isspace(p[-1]))
      *p++ = ' ';
        } *--p = 0;
    }
  else
      {
      strcpy(buf,"");
      while(fgets(str1,80,fp) != NULL)
      strcat(buf,str1);
      }
  /* description window */
  n=0; /* set options for the text window */
  XtSetArg(args[n],XmNrows, 60);n++;
  XtSetArg(args[n],XmNcolumns, 80);n++;
  XtSetArg(args[n],XmNeditable, False);n++;
  XtSetArg(args[n],XmNscrollHorizontal,False);n++;
  XtSetArg(args[n],XmNcursorPositionVisible,False);n++;
  XtSetArg(args[n],XmNwordWrap, True);n++;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNtopWidget,label); n++;
  XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNleftWidget,left_w); n++;
  XtSetArg(args[n],XmNvalue, buf); n++;
  text_w = XmCreateScrolledText(parent,"text_w",args,n);
  XtManageChild(text_w);
} /* end of create_description_w() */


/*
MakeXmString() creates Xmstring from char string.
*/

XmString MakeXmString(string)
char *string;
{
  return XmStringLtoRCreate(string,XmSTRING_DEFAULT_CHARSET);
} /* end of MakeXmString() */


/*
quit() closes display and terminate the program as a callback procedure.
*/
```

```
static void quit(w,main_w,call_data)
Widget w;
Widget main_w;
XmAnyCallbackStruct *call_data;
{
      XtCloseDisplay(XtDisplay(w));
      exit(0);
} /* end of quit() */



/*
show_fselbox() pops up the  file selection box as a callback procedure.
*/

static void show_fselbox(w,fselbox,call_data)
Widget w;  /* button[0] -- file button */
Widget fselbox; /* pointer to file selection box widget */
XmAnyCallbackStruct *call_data;
{
  XtManageChild(fselbox);
} /* end of show_fselbox() */


/*
create_fselbox() creates a file selection box which has a Motif-defined widet,
FileSelectionDialog. It sets default dir to /usr/adm/sa/ and returns a pointer to the box.
*/

Widget create_fselbox(parent)
Widget parent; /* button[0] */
{
  Widget fselbox;
  Arg args[NUM_ARGS];
  int n;
  XmString str,default_dir = MakeXmString("/usr/adm/sa/");

  str = MakeXmString("File Selection");
  n=0; /* set default dir and dialog title */
  XtSetArg(args[n],XmNdirMask,default_dir);n++;
  XtSetArg(args[n],XmNdialogTitle,str);n++;
  XtSetArg(args[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL); n++;
  XtSetArg(args[n],XmNhelpLabelString,
    MakeXmString("Today")); n++;  /* set help label to Today  */
  fselbox = XmCreateFileSelectionDialog(parent,"fselbox", args,n);
   /* register callback related to OK button in the box */
  XtAddCallback(fselbox,XmNokCallback,popdownFile,parent);
  /* register callback related to cancel button in the box */
  XtAddCallback(fselbox,XmNcancelCallback,fselbox_cancel,parent);
  /* register callback related to Today button in the box */
  XtAddCallback(fselbox,XmNhelpCallback,fselbox_default,parent) ;
  return (fselbox);
} /* end of create_fselbox() */


/*
popdown() pops down the widget window passed to it as a callback procedure and toggles
back the corresponding button.
*/

static void popdown(w,dia_box,call_data)
Widget w;
Widget dia_box;
XmAnyCallbackStruct *call_data;
{
  XtUnmanageChild(dia_box);
  XmToggleButtonSetState(w,False,False);
} /* end of popdown() */


/*
unset_state() toggles back the correspoding button.
*/

static void unset_state(w,bt,call_data)
Widget w;
Widget bt;
```

```
XmAnyCallbackStruct *call_data;
{
   w = XtParent(w); /* get parent widget */

   if(call_data->reason == XmCR_UNMAP)
   XmToggleButtonSetState(XtParent(w),False,False);
   else
   XmToggleButtonSetState(bt,False,False);
} /* end of unset_state() */


Widget text_w[3];
/*
create_opselbox() creates an options selection box in correspondance to the options
button. The box has widgets of BulletinBoardDialog, pane, form, push buttons and etc. It
returns a pointer to the box widget.
*/
Widget create_opselbox(parent)
Widget parent; /* button[1] -- the options button */
{
  Widget dia_shell,pane,opselbox,form,pb,ok_pb;
  Widget mainform, subform,label;
  char buf[20];
  Arg args[NUM_ARGS];
  int n,i;

  n=0; /* set style, title, and etc. for the box */
  XtSetArg(args[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL);n++;
  XtSetArg(args[n],XmNdialogTitle,MakeXmString("Option Selection"));n++;
  XtSetArg(args[n],XmNmarginWidth,1);n++;
  XtSetArg(args[n],XmNnoResize,True);n++; /* fix the dia_box size */
  dia_shell = XmCreateBulletinBoardDialog(parent,"bulletin",args,n);

  n=0; /* set pane's sash size */
  XtSetArg(args[n],XmNsashWidth,1); n++;
  XtSetArg(args[n],XmNsashHeight,1); n++;
  pane = XtCreateWidget("pane",xmPanedWindowWidgetClass,dia_shell,args,n);

  /* change char strings to XmString for box listing */
  for(i=0; i<XtNumber(List); i++)
    XmList[i] = MakeXmString(List[i]);

  n=0; /* set options for SscrolledList widget */
  XtSetArg(args[n],XmNitems,XmList); n++;
  XtSetArg(args[n],XmNitemCount,XtNumber(List));n++;
  XtSetArg(args[n],XmNscrollBarDisplayPolicy,XmAS_NEEDED); n++;
  XtSetArg(args[n],XmNselectionPolicy,XmMULTIPLE_SELECT);n++;
  XtSetArg(args[n],XmNvisibleItemCount,XtNumber(List));n++;
  opselbox = XmCreateScrolledList(pane,"opselbox",args,n);
  XtManageChild(opselbox);
  list_w = opselbox;

  n=0; /* create a form window with 5 fractions */
   XtSetArg(args[n],XmNfractionBase, 5); n++; form =
   XtCreateWidget("form",xmFormWidgetClass,pane,args,n);

  n=0; /* set OK button's position */
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION); n++;
  XtSetArg(args[n],XmNleftPosition,1); n++;
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_POSITION); n++ ;
  XtSetArg(args[n],XmNrightPosition,2); n++;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNshowAsDefault,True); n++;
  ok_pb = XtCreateManagedWidget("OK",xmPushButtonGadgetClass,form,args,n);

  /* register callbacks to OK button action */
  XtAddCallback(ok_pb,XmNactivateCallback,get_choices,dia_shell);
  XtAddCallback(ok_pb,XmNactivateCallback,unset_state,parent) ;

  n=0; /* set Cancel button's position */
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION); n++;
  XtSetArg(args[n],XmNleftPosition,3); n++;
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_POSITION); n++ ;
  XtSetArg(args[n],XmNrightPosition,4); n++;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
```

```
    XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
    XtSetArg(args[n],XmNshowAsDefault,True); n++;
    pb = XtCreateManagedWidget("Cancel",xmPushButtonGadgetClass,form,args,n);
    /* register callbacks to Cancel button action */
    XtAddCallback(pb,XmNactivateCallback,popdownbox,dia_shell);
    XtAddCallback(pb,XmNactivateCallback,unset_state,parent);

    XtManageChild(form);
    /* Fix the action area pane to its current height */
    {
    Dimension h;
      n=0;
      XtSetArg(args[n],XmNheight,&h); n++;
      XtGetValues(pb,args,n);

      n=0;
      XtSetArg(args[n],XmNpaneMinimum,h+10); n++;
      XtSetArg(args[n],XmNpaneMaximum,h+10); n++;
      XtSetValues(form,args,n);
    }
    n=0;
    XtSetArg(args[n],XmNshowAsDefault,False); n++;
    XtSetValues(pb,args,n);

    /* set options for time and interval */

    mainform = XtCreateWidget("mainform",xmFormWidgetClass,pane,args,0);
    for(i=0;i<XtNumber(prompts); i++)
      { /* set positions of options */
      n=0;
      XtSetArg(args[n],XmNtopAttachment,i? XmATTACH_WIDGET : XmATTACH_FORM );n++;
      XtSetArg(args[n],XmNtopWidget,subform); n++;
      XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM); n++;
      XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM); n++;

      subform = XtCreateWidget("subform",xmFormWidgetClass,mainform,args,n);

      n=0;
      XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
      XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM); n++;
      XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
      XtSetArg(args[n],XmNalignment,XmALIGNMENT_BEGINNING); n++ ;
      label = XtCreateManagedWidget(prompts[i],xmLabelGadgetClass,subform,args,n);

      sprintf(buf,"text_%d",i);

      n=0;
      XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
      XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
      XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM); n++;
      XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET); n++;
      XtSetArg(args[n],XmNleftWidget,label); n++;

      text_w[i] = XtCreateManagedWidget(buf,xmTextWidgetClass,subform,args,n);

      XtManageChild(subform);
      }
      XtManageChild(mainform);

  XtManageChild(pane);
  return (dia_shell);
} /* end of create_opselbox() */


/*
show_opselbox() pops up the options selection box as a callback procedure.
*/

static void show_opselbox(w,dia_shell,call_data)
Widget w; /* option button */
Widget dia_shell;
XmAnyCallbackStruct *call_data;
{
  XtManageChild(dia_shell);
} /* end of show_opselbox() */
```

```
/*
fselbox_default() handles the file selection default action as a callback procedure.
*/

void fselbox_default(w,filebt,call_data)
Widget w; Widget filebt;
XmAnyCallbackStruct *call_data;
{
  strcpy(choosen.filename,""); /* empty file name */
  /* toggle back the corresponding button */
  XmToggleButtonSetState(filebt,False,False);
  XtUnmanageChild(w);
} /* end of fselbox_default() */


/*
fselbox_cancel() handles events as a result of Cancel button
callback. It toggles back the corresponding button and pops down the corresponding window.
*/

void fselbox_cancel(w,filebt,call_data)
Widget w;
Widget filebt;
XmAnyCallbackStruct *call_data;
{
  XmToggleButtonSetState(filebt,False,False);
  XtUnmanageChild(w);
} /* end of fselbox_cancel() */


/*
popdownFile(), called as an OK buttton callback procedure,
checks for the choosen file name and the file type, and pop downs the file selection box.
In case of empty file name
and invalid file type, an error message box is popped up, and the user has to respond to
it before any further intercation can go.
*/

void popdownFile(w,filebt,call_data)
Widget w;
Widget filebt; /* pointer to file button */
XmAnyCallbackStruct *call_data;
{
  Widget sbox;
  Arg args[NUM_ARGS];
  XmString s;
  char *str,filename[80],filetype[80],cmd[150];
  int n,num;
  FILE *fp;

  /* get dialog text widget */
  sbox = XmFileSelectionBoxGetChild(w,XmDIALOG_TEXT);
  str = XmTextGetString(sbox); /* get file name string */
  if(strlen(str) <1) /* check string length */
    { /* file name not specified */
    if(!error_box) /* error box not created yet */
    error_box = create_error_box(w,"Invalid File Name");
    else /* error box already created */
    {
    n=0;
    XtSetArg(args[n],XmNmessageString,MakeXmString("Invalid File Name"));n++;
    XtSetValues(error_box,args,n);
    }
    XtManageChild(error_box);
    return;
    }
  /* set up a shell script to decide file type  */
  strcpy(cmd,"/bin/csh -fc 'file ");
   strcat(cmd,str); strcat(cmd," >/tmp/filetype");
  system(cmd);

  if( (fp = fopen("/tmp/filetype","r")) == NULL)
    { printf("\nCannot open /tmp/filetype\n"); exit(1); }

  fscanf(fp,"%s %s",filename,filetype);
  fclose(fp);
  system("rm -f /tmp/filetype");
```

```
  if(strncmp(filetype,"data",4)!= 0) /* not data type file */
    {
    if(!error_box)
    error_box = create_error_box(w,"Invalid File\nMust be Data Type");
    else
    {
    n=0;
    XtSetArg(args[n],XmNmessageString,
       MakeXmString("Invalid File\nMust be Data Type")); n++;
    XtSetValues(error_box,args,n);
    }
    XtManageChild(error_box);
    return;
    }

  strcpy(choosen.filename,str); /* store file name */
  XtFree(str);
  XmToggleButtonSetState(filebt,False,False);
  /* state, notify */
  XtUnmanageChild(w);
} /* end of popdownFile() */


/*
get_strings() gets options for interval and time in the fileselection box.
*/

void get_strings()
{
  int i;
  char *text;

  text = XmTextGetString(text_w[0]); /* get start time */
  if(strcmp(text,"")!=0)
    { /* not empty string */
      sprintf(choosen.start,"-s %s ",text); /* store start time */
      XmTextSetString(text_w[0],""); /* set start time to empty */
      XtFree(text);
      }
  else /* empty string */
      strcpy(choosen.start,"");

  text = XmTextGetString(text_w[1]); /* get end time */
  if(strcmp(text,"")!=0)
      { /* not empty string */
      sprintf(choosen.end,"-e %s ",text); /* store end time */
      XmTextSetString(text_w[1],""); /* set end time to empty */
      XtFree(text);
      }
  else
      strcpy(choosen.end,"");

  text = XmTextGetString(text_w[2]); /* get interval */
  if(strcmp(text,"")!=0)
      {
      sprintf(choosen.interval,"-i %s ",text);
      XmTextSetString(text_w[2],"");
      XtFree(text);
      }
  else
      strcpy(choosen.interval,"");
} /* end of get_strings() */


/*
get_choices(), a callback procedure, obtains item options and options for start and end
time and for interval. If no item is selected, an error message box is popped up and the
user must respond to it before any interaction can go on. It pops down the options box and
calls the get_sar_data function. If the data read-in fails, error message is displayed
through
a popped up box.
*/

static void get_choices(w,dia_shell,call_data)
Widget w;Widget dia_shell;
XmAnyCallbackStruct *call_data;
{
```

```
    Widget ancestor = XtParent(XtParent(dia_shell));
    Arg args[2];
    int n,i,j,tabel_status=(-1);
    int *pos_list; /* list of items in terms of position */
    int pos_cnt; /* number of items selected */

    if(!XmListGetSelectedPos(list_w,&pos_list,&pos_cnt))
        { /* no item selected */
        if(!error_box)
          error_box = create_error_box(ancestor,"NO Option Has Been Choosen.");
        else
          {
          n=0;
          XtSetArg(args[n],XmNmessageString,
            MakeXmString("NO Option Has Been Choosen.")); n++;
          XtSetValues(error_box,args,n);
          }
        XtManageChild(error_box);
        return;
        }

    get_strings(); /* get options for start and end time and */
                   /* for interval */
    choosen.num = pos_cnt; /* store number of items choosen */
    for(i=0; i<pos_cnt; i++)
        { /* store position list into global var */
        if(pos_list[i] == 9)
          { /* set position 9 item into last for read-in reason */
          pos_list[i] = pos_list[pos_cnt-1];
          pos_list[pos_cnt-1] = 9;
          }
        choosen.pos[i] = pos_list[i];
        }
    XtFree(pos_list);

    /* pop down the file selection box */
    XtUnmanageChild(dia_shell);
    /* get sar data from the choosen file */
    if(get_sar_data() ==1)
        { /* fail to get data from the file */
        if(!error_box)
          error_box = create_error_box(ancestor,"Invalid File\nFormat Unmatched");
        else
          {
          n=0;
          XtSetArg(args[n],XmNmessageString,
            MakeXmString("Invalid File\nFormat Unmatched")); n++;
          XtSetValues(error_box,args,n);
          }
        XtManageChild(error_box);
        }
} /* end of get_choices() */


/*
create_error_box() creates an error message box which has
a Motif-defined widget, ErrorDialog. It returns a pointer to the error box.
*/

Widget create_error_box(parent,message)
Widget parent;
char *message;
{
  Widget dia_box,bt;
  Arg args[NUM_ARGS];
  int n;
  XmString str;

  /* change char string to XmString */
  str = MakeXmString(message);

  n=0; /* set dialog style and error message */
  XtSetArg(args[n],XmNdialogStyle,XmDIALOG_PRIMARY_APPLICATION_MODAL);n++;
  XtSetArg(args[n],XmNmessageString,str); n++;
  dia_box = XmCreateErrorDialog(parent,"error",args,n);
  /* unmanage unused buttons */
  XtUnmanageChild(XmMessageBoxGetChild(dia_box,XmDIALOG_CANCEL_BUTTON));
```

```
  XtUnmanageChild(XmMessageBoxGetChild(dia_box,XmDIALOG_HELP_BUTTON));
  /* register callback for OK button */
  XtAddCallback(dia_box,XmNokCallback,popdownbox,dia_box);
  return (dia_box);
} /* end of create_error_box() */


/*
popdownbox() pops down a window passed to it.
*/

void popdownbox(w,dia_box,call_data)
Widget w;
Widget dia_box;
XmAnyCallbackStruct *call_data;
{
   XtUnmanageChild(dia_box);
} /* end of popdownbox() */


/*
display_working_cursor() checks for toggle button state and displays the working cursor
accordingly. If the toggle button
has been already selected, it will reset the button state and returns. Otherwise, it will
display the cursor on the screen.
*/

void display_working_cursor(w,form,call_data)
Widget w;
Widget form;
XmToggleButtonCallbackStruct *call_data;
{
  if(call_data->set == False)
      {
      XmToggleButtonSetState(w,True,False);
      return;
      }
  if(form) /* display the cursor in the form window */
    XDefineCursor(XtDisplay(w),XtWindow(form),working_cursor);
  else /* display it in the graphics box */

  XDefineCursor(XtDisplay(w),XtWindow(graph_box_globe),working_cursor);
} /* end of display_working_cursor() */


/*
create_working_cursor() creates a watch symbol to indicate the working cursor, and returns
a pointer to the cursor.
*/

Cursor create_working_cursor(w)
Widget w;
{
  return XCreateFontCursor(XtDisplay(w),XC_watch);
} /* end of create_working_cursor() */


/*
undisplay_working_cursor() sets the cursor to its default
symbol.
*/

void undisplay_working_cursor(w,whichone,call_data)
Widget w;
int whichone;
XmAnyCallbackStruct *call_data;
{
  XUndefineCursor(XtDisplay(w),XtWindow(graph_box_globe));
} /* end of undisplay_working_cursor() */

/*
init_sa() initializes the application parameters.
*/

init_sa()
{
  int i;
```

```
  sa.cpu_u = NULL; sa.buf = NULL;  sa.blk_dev = NULL;
  sa.tty = NULL;   sa.sysc = NULL; sa.syss=NULL;
  sa.filea=NULL;   sa.q = NULL;    sa.tbs = NULL;
  sa.msg = NULL;   sa.p = NULL;    sa.mem_blk = NULL;
  sa.y_axis = NULL;

  for(i=0;i<12;i++)
      {
      draw_para[i].window = NULL;
      draw_para[i].pixmap = NULL;
      }
  for(i=0; i<14; i++)  choosen.positions[i] = 0;
  choosen.num = 0;
  strcpy(choosen.filename,"");
} /* end of init_sa() */


/*
clean_sa() returns the memories pointed to by Sa
*/

clean_sa()
{
  if(sa.cpu_u)
      {
      free(sa.cpu_u);
      sa.cpu_u = NULL;
      }
  if(sa.buf)
      { free(sa.buf);
      sa.buf = NULL;
      }
  if(sa.blk_dev)
      {
      free(sa.blk_dev);
      sa.blk_dev = NULL;
      }
  if(sa.tty)
      {
      free(sa.tty);
      sa.tty = NULL;
      }
  if(sa.sysc)
      {
      free(sa.sysc);
      sa.sysc = NULL;
      }
  if(sa.syss)
      {
      free(sa.syss);
      sa.syss=NULL;
      }
  if(sa.filea)
      {
      free(sa.filea);
      sa.filea=NULL;
      }
  if(sa.q)
      {
      free(sa.q);
      sa.q = NULL;
      }
  if(sa.tbs)
      {
      free(sa.tbs);    sa.tbs = NULL;
      }
  if(sa.msg)
      {
      free(sa.msg);
      sa.msg = NULL;
      }
  if(sa.p)
      {
      free(sa.p);
      sa.p = NULL;
      }
  if(sa.mem_blk)
```

```
        {
        free(sa.mem_blk);
        sa.mem_blk = NULL;
        }
   if(sa.y_axis) {
        free(sa.y_axis);
        sa.y_axis = NULL;
        }
} /* end of clean_sa() */


/*
The des.text starts here
*/

     This program provides a user interface for graphical display of the Sequent S/81
system performance based on data obtained by the sar command. For data details, please
refer to the MAN page for sar.

     The default directory is set to /usr/adm/sa/, where DYNIX/ptx keeps its daily file of
system activity data in binary format. You can, however, specify any file as long as it is
generated as a result of running sar.

     The programmer, Haibo Du, welcomes any suggestion or comments about this program.

     Email: dhaibo@a.cs.okstate.edu



/* graph.h starts here */
/* variables and type declarations for graph.c */
#ifndef H_GRAPH
#define H_GRAPH

#define fg_width 16
#define fg_height 16
static unsigned char fg_bitmap[32] = {/* solid foreground */
  0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
  0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
  0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
  0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
};

#define cross_weave_width 16
#define cross_weave_height 16
static char cross_weave_bits[] = { /* weave pattern */
  0x55,0x55,0x88,0x88,0x55,0x55,0x22,0x22,
  0x55,0x55,0x88,0x88,0x55 0x55,0x22,0x22,
  0x55,0x55,0x88,0x88,0x55,0x55,0x22,0x22,
  0x55,0x55,0x88,0x88,0x55,0x55,0x22,0x22};

static char *patterns[] = { /* tile pattern used in drawing pie and bar charts */
  "foreground",
  "cross_weave",
  "25_foreground",
  "50_foreground",
  "75_foreground",
  "vertical",
  "horizontal",
  "slant_right",
  "slant_left",
  "background",
};

#include <Xm/DialogS.h>
#include <Xm/PanedW.h>
#include <Xm/MessageB.h>

#define Yes 1
#define No  0
#define absa(i)  (i)<0 ? -(i) : (i) /* get absolute value */
#define CPU_U   0
#define BUF     1
#define BLK_DEV 2
#define TTY_    3
```

```
#define SYSC    4
#define SYSS    5
#define FILEA   6
#define Q       7
#define TBS     8
#define MSG     9
#define P       10
#define MEM_BLK 11
#define X_START 117 /* start position for x axis */

void redraw(),clear_it(),draw_cpu_cb(),draw_buf_cb();
void draw_blk_dev_cb(),draw_tty_cb(),draw_sysc_cb();
void draw_syss_cb(),draw_filea_cb(),draw_q_cb();
void draw_tbs_cb(),draw_msg_cb(),draw_p_cb();
void draw_mem_blk_cb(),graph_callback();
void make_legend(),quit_callback(),unset_state_2();
void save_cb(), save_map_cb(), do_xwd(), save_xwdfile();
GC create_gc();
XmString MakeXmString();
long get_display_length();
Pixmap create_pixmap();
Widget create_drawing_w(),create_form_window();
Widget create_frame_window(),create_sys_label();
Widget create_scrolled_window(),create_canvas_window();
Widget create_legend_window(),create_help_box();
Widget get_form_from_canvas();
void create_draw_menu(),create_draw_env();draw_rectangle();
void draw_scaler(),draw_time(),set_sys_name(),to_ascii();
void str_rev(),delete_pixmap();
void delete_pixmap_cb(),create_device_menu(),
void change_device_menu(),unmap_dev_cmd_cb(),help_cb();
void display_working_cursor(),undisplay_working_cursor();
int reset_max_int(),reset_max_float();

static Widget sys_label,device_bt[30],Device_Menu_Command;
static Widget Device_Canvas,help_box=NULL;

/* globel variables */
extern Sa sa;
extern Widget graph_box_globe;
extern Cursor working_cursor;
extern Drawing_Para draw_para[];

#endif



/* graph.c starts here */
#include "sarview.h"
#include "motif.h"
#include "graph.h"

static Widget sys_labels[12]; /* information on system host name and others */

/*
graph_callback() handles actions required as a result of a press of a corresponding toggle
button. It first checks to see if the toggle button has already been pressed. Then it
creates a help box and a corresponding drawing window if they have not been created.
Finally, it pops up the corresponding drawing window.
*/

void graph_callback(w,whichone,call_data)
Widget w;
int whichone; /* indix to a specific option */
XmToggleButtonCallbackStruct *call_data;
{
  char title[80];
  int graph_num,i;

  if(call_data->set == False)
    { /* press the toggle button when it has been pressed */
    XmToggleButtonSetState(w,True,False);
    return;
    }
  if(!help_box)
       help_box = create_help_box(w); /* create a help box */
  if(!draw_para[whichone].window)
```

```
    { /* the corresponding drawing window not created */
    switch(whichone)
        { /* decide its title and bar chart number */
        case 0: strcpy(title,"CPU Utlization");
                graph_num = 3; /* 3 bar charts */
                break;
        case 1: strcpy(title,"Buffer Activity");
                graph_num = 4;
                break;
        case 2: strcpy(title,"Block Device Activity");
                graph_num = 3;
                break;
        case 3: strcpy(title,"TTY Device Activity");
                graph_num = 4;
                break;
        case 4: strcpy(title,"System Calls");
                graph_num = 4;
                break;
        case 5: strcpy(title,"System Swapping & Switching");
                graph_num = 3;
                break;
        case 6: strcpy(title,"File Access System Routine Usage");
                graph_num = 2;
                break;
        case 7: strcpy(title,"Average Queue Length");
                graph_num = 2;
                break;
        case 8: strcpy(title,"Status of Process, Inode,File Tables");
                graph_num = 4;
                break;
        case 9: strcpy(title,"Message & Semaphore Activities");
                graph_num = 1;
                break;
       case 10: strcpy(title,"Paging Activities");
                graph_num = 2;
                break;
       case 11: strcpy(title,"Unused Memory Pages & Disk Blocks");
                graph_num = 1;
                break;
        default: printf("\nIllegal whichone %d in graph_callback \n",whichone); exit(-1);
        }
    /* create a corresponding drawing window */
    draw_para[whichone].window= create_drawing_w(w,title,
         whichone,graph_num);
    }
  else /* the corresponding drawing window already created */
    {
    if(whichone == 2) /* blcok device activity */
        change_device_menu();
    /* reset system/host name and date of data collected */
    set_sys_name(draw_para[whichone].window,whichone);
    }
  XtManageChild(draw_para[whichone].window);
} /* end of graph_callback() */


/*
set_sys_name() updates the system name and date of data. It
also sets the default window size to ensure the required display window size.
*/

void set_sys_name(w,whichone)
Widget w;
int whichone;
{
  int n;
  Arg args[10];
  XmString str;

  str = MakeXmString(sa.sys_name);
  n=0;
  XtSetArg(args[n],XmNlabelString,str); n++;
  XtSetValues(sys_labels[whichone],args,n);
  XFree(str);
  n=0;
  XtSetArg(args[n],XmNwidth,726); n++;
```

```
  XtSetArg(args[n],XmNheight,670);n++;
  XtSetValues(w,args,n);
}  /* end of set_sys_name() */


/*
create_drawing_w() creates a drawing window to display the corresponding data in bar chart
or pie chart. The window has widgets of Motif-defined FormDialog, frame, rowcolumn, toggle
button, label, drawing. It returns a pointer to the drawing window.
*/

Widget create_drawing_w(w,title,whichone,graph_num)
Widget w;
char *title;
int whichone,graph_num;
{
  Widget canvas,form,shell,command,tiles,sep,frame,sw,legend;
  Widget pb,rc,pie_bt,bar_bt,quit,label; XGCValues gcv;
  int i,n,id,pos,scr;
  Arg args[10];
  Widget xs_create_pixmap_browser();
  Pixel fg,bg;
  Pixmap tile,pixmap;
  Display *dpy;
  XmString str;
  GC gc;
  Dimension width,height;
  static int register_pattern = No;

  /* create a FormDialog widget */
  form = create_form_window(w,title);
  /* create a frame widget for display menu */
  frame = create_frame_window(form);

  /* Create the row column widget to hold the commands */
  command = XtCreateManagedWidget("command",xmRowColumnWidgetClass,frame,NULL,0);

  /* create host name  label */
  sys_label = create_sys_label(form,frame);
  sys_labels[whichone] = sys_label;
  /* create scrolled window */
  sw = create_scrolled_window(form,frame);

  if(whichone == 0)  /* cpu_utilization */
    canvas = create_canvas_window(sw,whichone,850,
      graph_num*840); /* wider than the other display windows */
  else
   canvas = create_canvas_window(sw,whichone,726,graph_num*840);

  create_draw_menu(command,form,w,canvas,whichone);

  /* add a palette of fill patterns. */
  if(register_pattern == No) /* not yet added */
    {
    xs_register_pattern(form,"foreground",fg_bitmap,fg_width,fg_height); /* black color */
    xs_register_pattern(form,"cross_weave",
    cross_weave_bits,cross_weave_width,
    cross_weave_height); /* cross_weave pattern */
    register_pattern = Yes;
    }

  /* make legend */
  legend = create_legend_window(frame,form,sw,whichone,80);

  n=0;
  XtSetArg(args[n],XmNunitType,XmPIXELS); n++;
  XtSetValues(canvas,args,n);

  switch (whichone)
    { /* register map callback  accordingly */
    case 0: XtAddCallback(form,XmNmapCallback,undisplay_working_cursor,whichone);
            break; /* indirect drawing */
    case 1: XtAddCallback(form,XmNmapCallback,draw_buf_cb,canvas);
           break; /* direct drawing */
    case 2: XtAddCallback(form,XmNmapCallback,undisplay_working_cursor,whichone);
           break; /* indirect drawing */
    case 3: XtAddCallback(form,XmNmapCallback,draw_tty_cb,canvas);
```

```
            break; /* the rest are all direct drawing */
    case 4: XtAddCallback(form,XmNmapCallback,draw_sysc_cb,canvas);
            break;
    case 5: XtAddCallback(form,XmNmapCallback,draw_syss_cb,canvas);
            break;
    case 6: XtAddCallback(form,XmNmapCallback,draw_filea_cb,canvas);
            break;
    case 7: XtAddCallback(form,XmNmapCallback,draw_q_cb,canvas);
            break;
    case 8: XtAddCallback(form,XmNmapCallback,draw_tbs_cb,canvas);
            break;
    case 9: XtAddCallback(form,XmNmapCallback,draw_msg_cb,canvas);
             break;
    case 10: XtAddCallback(form,XmNmapCallback,draw_p_cb,canvas);
             break;
    case 11: XtAddCallback(form,XmNmapCallback,draw_mem_blk_cb,canvas);
             break;
    default: printf("\nIllegal whichone %d in create_drawing_w\n",whichone); exit (-1);
    }
  /* register callback when unmapped */
  XtAddCallback(form,XmNunmapCallback,
    delete_pixmap_cb,whichone);
  return (form);
} /* end of create_drawing_w() */


/*
create_pixmap() creates a pixmap the same size of the drawing window passed to it. It also
create graphic content used to draw into the pixmap.
*/

Pixmap create_pixmap(canvas,whichone)
Widget canvas;
int whichone;
{
  int n,x,y;
  Arg args[10];
  Dimension width,height;
  Pixmap pixmap;
  GC gc;
  XGCValues gcv;
  unsigned int w,h,bdw,depth;

  n=0; /* get the size of the canvas */
  XtSetArg(args[n],XmNwidth,&width); n++;
  XtSetArg(args[n],XmNheight,&height); n++;
  XtGetValues(canvas,args,n);

  /* create a pixmap the same size as the drawing area. */
  while(1)
    {
    pixmap = XCreatePixmap(XtDisplay(canvas),
      RootWindowOfScreen(XtScreen(canvas)),width,
      height,DefaultDepthOfScreen(XtScreen(canvas)));

    if(!XGetGeometry(XtDisplay(canvas),pixmap,
      &RootWindowOfScreen(XtScreen(canvas)),
      &x,&y,&w,&h,&bdw,&depth))
      /* fail to create pixmap */
      delete_pixmap(XtDisplay(canvas));
    else
      break;
    }
  /* save the size of the particular pixmap */
  draw_para[whichone].width = width;
  draw_para[whichone].height = height;

  /* Create a GC for drawing (callback) */
  gcv.foreground = WhitePixelOfScreen(XtScreen(canvas));
  gc = XCreateGC(XtDisplay(canvas),RootWindowOfScreen(XtScreen(canvas)),
   GCForeground,&gcv);

 /* Create Pixmap with white */
  XFillRectangle(XtDisplay(canvas),pixmap,gc,0,0,width,height);
  /* drawing is now drawn into with "black"; change the gc */
  XSetForeground(XtDisplay(canvas),gc,BlackPixelOfScreen(XtScreen(canvas)));
  draw_para[whichone].gc = gc; /* save the gc */
```

```
  return pixmap;
} /* end of create_pixmap() */


/*
delete_pixmap_cb() frees the data structure associated with
the particular pixmap.
*/

void delete_pixmap_cb(w,whichone,cbs)
Widget w;
int whichone;
XmAnyCallbackStruct *cbs;
{
  int n,i;
  Arg args[10];

  if(!draw_para[whichone].pixmap) /* Null pointer */
    return;
  XFreePixmap(XtDisplay(w),draw_para[whichone].pixmap);
  draw_para[whichone].pixmap = NULL;
} /* end of delete_pixmap_cb() */


/*
delete_pixmap() goes through the pixmap array and set free a pixmap if it finds one.
*/

void delete_pixmap(dpy)
Display *dpy;
{
  int i;

  for(i=0;i<12;i++)
    {
    if(draw_para[i].pixmap)
      {
      XFreePixmap(dpy,draw_para[i].pixmap);
      draw_para[i].pixmap = NULL;
      return;
      }
    }
  printf("\nError: not enough memory for pixmap\n");
  exit(-1);
} /* end of delete_pixmap */


/*
create_form_window() creates a form window by calling a Motif-specified function,
XmCreateFormDialog(). The form window is the container for other windows in the drawing
window. It returns a pointer to the form widget.
*/

Widget create_form_window(parent,title)
Widget parent;
char *title;
{
  Widget form;
  int n;
  Arg args[10];

  n=0; /* set title and size */
  XtSetArg(args[n],XmNdialogTitle,MakeXmString(title));n++;
  XtSetArg(args[n],XmNwidth,726); n++;
  XtSetArg(args[n],XmNheight,670);n++;
  form = XmCreateFormDialog(parent,"form",args,n);

  return form;
} /* end of create_form_window() */


/*
create_frame_window() creates a frame widget placed at the particular position in the form
widget and returns a pointer
to the frame widget.
*/
```

```
Widget create_frame_window(parent)
Widget parent;
{
  int n;
  Arg args[10];

  n=0; /* set geometry and shadow type */
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM);n++;
  XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM);n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM);n++;
  XtSetArg(args[n],XmNshadowType,XmSHADOW_IN);n++;

  return XtCreateManagedWidget("frame",xmFrameWidgetClass,parent,args,n);
} /* end of create_frame_window() */


/*
create_sys_label() creates a label widget for the system name and data date. It returns a
pointer to the label widget.
*/

Widget create_sys_label(form,frame)
Widget form,frame;
{
  int n;
  Arg args[10];
  XmString str;
  Widget label;

  str = MakeXmString(sa.sys_name);
  n=0; /* set geometry */
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNleftWidget,frame); n++;
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNlabelString,str);n++;
  XtSetArg(args[n],XmNx,70);n++;
  label = XtCreateManagedWidget("label",xmLabelWidgetClass,form,args,n);
  XmStringFree(str);

  return label;
} /* end of create_sys_label() */


/*
create_scrolled_window() creates a scrolled widget, which is used to hold the drawing
widget to make the scrolling of the drawing window possible. It returns a pointer to the
scrolled widget.
*/

Widget create_scrolled_window(form,frame)
Widget form,frame;
{
  int n;
  Arg args[10];

  n=0;/* set geometry, size and policy */
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNtopWidget,sys_label); n++;
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNleftWidget,frame); n++;
  XtSetArg(args[n],XmNwidth,300); n++;
  XtSetArg(args[n],XmNscrollingPolicy,XmAUTOMATIC); n++;
  XtSetArg(args[n],XmNscrollBarDisplayPolicy,XmAS_NEEDED);n++ ;
  return XtCreateManagedWidget("scrolled_win",
        xmScrolledWindowWidgetClass,form,args,n);
} /* end of create_scrolled_window() */


/*
create_canvas_window() creates a drawing area widget used for
pie and bar chart drawing. It returns a pointer to the drawing widget.
*/

Widget create_canvas_window(parent,whichone,width,length)
Widget parent;
```

```
int whichone,width,length; {
  Widget canvas;
  int n;
  Arg args[10];

  n=0; /* set size, resize policy, and user data */
  XtSetArg(args[n],XmNwidth, width); n++;
  XtSetArg(args[n],XmNheight,length); n++;
  XtSetArg(args[n],XmNresizePolicy, XmRESIZE_NONE);n++;
  XtSetArg(args[n],XmNuserData,whichone);n++;
  canvas = XtCreateManagedWidget("canvas",
          xmDrawingAreaWidgetClass,parent,args,n);
  /* register callback corresponding to the exposure */
  XtAddCallback(canvas,XmNexposeCallback,redraw,whichone);
  if(whichone == 2)
    Device_Canvas = canvas;

  return canvas;
} /* end of create_canvas_window() */


/*
create_legend_window() creates a legend window according to the particular window passed
to it. It returns a pointer to
the legend window widget.
*/

Widget create_legend_window(frame,form,sw,whichone,fractions)
Widget frame,form,sw;
int whichone,fractions;
{
  Widget legend,label;
  Pixel fg,bg;
  int n,pos,i,num;
  Arg args[10];
  char names[10][20];

  n=0; /* set geometry and fraction base */
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNleftWidget,frame); n++;
  XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNfractionBase, fractions); n++;
  legend = XtCreateManagedWidget("legend",
          xmFormWidgetClass,form,args,n);
  n=0;
  XtSetArg(args[n],XmNbottomAttachment,XmATTACH_WIDGET);n++;
  XtSetArg(args[n],XmNbottomWidget,legend);n++;
  XtSetValues(sw,args,n);

  n=0; /* set position */
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION);n++;
  XtSetArg(args[n],XmNleftPosition,0); n++;
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_POSITION); n++ ;
  XtSetArg(args[n],XmNrightPosition,6); n++;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNlabelString,MakeXmString("Legend:"));n++;
  label = XtCreateManagedWidget("legend",xmLabelWidgetClass,legend,args,n);
  n=0; /* get forground and background colors */
  XtSetArg(args[n],XmNforeground,&fg); n++;
  XtSetArg(args[n],XmNbackground,&bg); n++;
  XtGetValues(legend,args,n);

  switch (whichone) {  /* decide which legend to create */
    case 0: create_legend(legend,6,"%usr",patterns[0],fg,bg);
            create_legend(legend,14,"%sys",patterns[4],fg,bg);
            create_legend(legend,22,"%wio",patterns[2],fg,bg);
            create_legend(legend,30,"%idle",patterns[8],fg,bg) ;
            break;
    case 1: num = 9; /* set legend item number */
            for(i=1;i<num;i++)
              strcpy(names[i],sa.buf_legend[i]);
            break;
    case 2: num = 7;
            for(i=1;i<num;i++)
              strcpy(names[i],sa.blk_legend[i+1]);
```

```
            break;
    case 3: num = 7;
            for(i=1;i<num;i++)
              strcpy(names[i],sa.tty_legend[i]);
            break;
    case 4: num = 8;
            for(i=1;i<num;i++)
              strcpy(names[i],sa.sysc_legend[i]);
            strcpy(names[i],"Negtive");
            num = 9;
            break;
    case 5: num = 6;
            for(i=1;i<num;i++)
              strcpy(names[i],sa.syss_legend[i]);
            break;
    case 6: num = 4;
            for(i=1;i<num;i++)
              strcpy(names[i],sa.filea_legend[i]);
            break;
    case 7: num = 5;
            for(i=1;i<num;i++)
              strcpy(names[i],sa.q_legend[i]);
            break;
    case 8: num = 8;
            for(i=1;i<num;i++)
              strcpy(names[i],sa.tbs_legend[i]);
            break;
    case 9: num = 3;
            for(i=1;i<num;i++)
              strcpy(names[i],sa.msg_legend[i]);
            break;
    case 10: num = 5;
             for(i=1;i<num;i++)
               strcpy(names[i],sa.p_legend[i]);
             break; case 11: num = 3;
             for(i=1;i<num;i++)
               strcpy(names[i],sa.mem_blk_legend[i]);
             break;
    default: printf("\nIllegal whichone %d in create legend \n",whichone);
               exit(-1);
    }
  if(whichone != 0) /* other than UPC Utilization */
        {
        pos=6;
        for(i=1;i<num;i++)
          {
          create_legend(legend,pos,names[i],
              patterns[i-1],fg,bg);
          pos += 8;
          }
        }
  return legend;
} /* end of create_legend_window() */


/*
create_draw_menu() creates button widgets with the drawing menu window. The buttons
include quit, save, and help. For CPU utilization and device activity, more buttons are
added.
*/

void create_draw_menu(command,form,w,canvas,whichone)
Widget command,form,w,canvas;
int whichone;
{
  Widget bt;
  int n,id;
  Arg args[10];
  /* create quit button */
  bt = XtCreateManagedWidget("quit",xmPushButtonWidgetClass,
        command,args,0);
  /* register callbacks related to the quit button */
  XtAddCallback(bt,XmNactivateCallback,quit_callback,form);
  XtAddCallback(bt,XmNactivateCallback,unset_state_2,w);

  /* create save button */
  bt = XtCreateManagedWidget("Save",
```

```
        xmPushButtonWidgetClass,command,args,0);
  /* register callbacks related to the save button */
  XtAddCallback(bt,XmNactivateCallback,save_cb,whichone);
  /* create help button and register related callback */
  bt = XtCreateManagedWidget("Help",
          xmPushButtonWidgetClass,command,args,0);
  XtAddCallback(bt,XmNactivateCallback,help_cb,whichone);

  if(whichone==0) /* for cpu utilization */
      { /* add pie and bar chart drawing buttons */
      n=0;
      XtSetArg(args[n],XmNuserData,0); n++;
      bt = XtCreateManagedWidget("Pie",
           xmPushButtonWidgetClass,command,args,n);
      XtAddCallback(bt,XmNactivateCallback,
        display_working_cursor,form);
      XtAddCallback(bt,XmNactivateCallback,draw_cpu_cb,canvas);
      n=0;
      XtSetArg(args[n],XmNuserData,1); n++;
      bt = XtCreateManagedWidget("Bar",
           xmPushButtonWidgetClass,command,args,n);
      XtAddCallback(bt,XmNactivateCallback,
        display_working_cursor,form);
      XtAddCallback(bt,XmNactivateCallback,draw_cpu_cb,canvas);
      }
  if(whichone == 2) /* for device activity */
      create_device_menu(command,canvas,whichone);
} /* end of create_draw_menu() */


/*
change_device_menu() checks the number of existing menu entries. If it is equal to the
needed one, it will do nothing and return. Otherwise it will either add more entries or
delete those not needed according to the difference.
*/

void change_device_menu()
{
  Widget w = Device_Menu_Command;
  Block_Device *blk_dev;
  int need,x,i,n,avail,diff;
  Arg args[10];
  XmString str;

  x = sa.index -1;
  blk_dev = sa.blk_dev;
  need = blk_dev[x].dev_no;
  if(device_bt_num == need) /* menu items same as before */
      return;

  diff = need - device_bt_num;
  if(diff>0) /* divice # fewer than needed */
      { /* create more device buttons */
      for(i=device_bt_num;i<need;i++)
        {
        n=0;
        XtSetArg(args[n],XmNuserData,i); n++;
        device_bt[i]= XtCreateManagedWidget
          (blk_dev[x].data[i].device,
          xmPushButtonWidgetClass,w,args,n);
        XtAddCallback(device_bt[i],XmNactivateCallback,
          draw_blk_dev_cb,Device_Canvas);
        }
      device_bt_num = i;
      }
  else /* delete extra device buttons */
      {
      for(i=need; i<device_bt_num;i++)
        XtDestroyWidget(device_bt[i]);
      device_bt_num = need;
      }
} /* end of change_device_menu */


/*
save_cb(), as a callback procedure, creates a prompt box of PromptDialog type if not yet
created, and then pops up that save box for prompting the user to enter the file name for
```

```
saving the specified drawing window.
*/

int thisone; /* a global var for later reference */
void save_cb(w,whichone,cbs)
Widget w;
int whichone;
XmAnyCallbackStruct *cbs;
{
  Widget rc,pb;
  Widget static save_box=NULL; /* create the prompt box only once */
  Arg    args[10];
  int n;
  XmString str;
  Position x,y;

  thisone = whichone;
  if(!save_box) /* save box not created yet */
    { /* create one */
  str = MakeXmString("Use xwud to view \nthe saved file\nEnter File Name:");
  n=0; /* set message string in the box  and other options */
  XtSetArg(args[n],XmNselectionLabelString,str); n++;
  XtSetArg(args[n],XmNautoUnmanage,False); n++;
  XtSetArg(args[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL);n++;
  XtSetArg(args[n],XmNallowOverlap,False);n++;
  XtSetArg(args[n],XmNdefaultPosition,False); n++;
  save_box = XmCreatePromptDialog(graph_box_globe,"prompt",args,n);
  /* register callback procedures associated with the save box */
  XtAddCallback(save_box,XmNokCallback,do_xwd,save_box);
  XtAddCallback(save_box,XmNcancelCallback,do_xwd,save_box);
  XtAddCallback(save_box,XmNmapCallback,save_box_map_cb,whichone);
  /* unmanage the unwanted widget */
  XtUnmanageChild(XmSelectionBoxGetChild(save_box,XmDIALOG_HELP_BUTTON));
  }
  XtManageChild(save_box);
} /* end of save_cb() */



/*
do_xwd() first checks the reason for the callback. If it is the result of cancel action,
it simply set the entry text to null and returns. Otherwise, it gets the file name
specified by the user. If it is empty, an error box will be created if not existing, and
popped up to echo such a error. When a file name is not empty, it updates the screen and
calls xwd_savefile to dump the window image into that file.
*/

char *filename=NULL; /* default */
char *frame="-frame";
void do_xwd(w,sb,cbs)
Widget w; Widget sb;
XmSelectionBoxCallbackStruct *cbs;
{
  Arg args[5];
  int n,i;
  static Widget xwd_error=NULL;
  extern Widget create_error_box();
  extern Widget graph_box_globe;
  Widget text;
  /* get the file name specified by the user */
  text = XmSelectionBoxGetChild(w,XmDIALOG_TEXT);

  switch (cbs->reason)
    { /* check for callback reason */
    case XmCR_CANCEL: /* as a result of pressing cancel button */
      XmTextSetString(text,""); /* set to null */
      XtUnmanageChild(w);
      break;
    case XmCR_OK: /* as a result of OK button */
      /* convert Xmstring to char string */
      XmStringGetLtoR(cbs->value,XmSTRING_DEFAULT_CHARSET, &filename);
      if(!filename) /* empty file name */
        {
        if(!xwd_error) /* error box not yet created */
          { /* create one */
          xwd_error = create_error_box(graph_box_globe,
            "File Name Is Not Specified!");
```

```
          n=0;
          XtSetArg(args[n],XmNautoUnmanage,False); n++;
          XtSetArg(args[n],XmNdialogStyle, XmDIALOG_SYSTEM_MODAL);n++;
          XtSetValues(xwd_error,args,n);
          }
        XtManageChild(xwd_error); /* pops up the box */
        return;
        }
      XmTextSetString(text,""); /* reset to null char */
      XtUnmanageChild(w);
      XmUpdateDisplay(w); /* make the prompt box popped down quickly */
      system("sleep 1"); /* allow time for server to act */
      save_xwdfile(w);
      break;
    default: printf("\n invalid callback value\n"); exit(1);
             break;
    } /* switch */
} /* end of do_xwd() */


/*
save_box_map_cb(), as a callback procedure, adjusts its geometry on the screen according
to the position of that particular drawing window which popps it up.
*/

void save_box_map_cb(w,whichone,cbs)
Widget w;
int whichone;
XmAnyCallbackStruct *cbs;
{
   Position x,y;
   Arg args[10];
   int n;

   n=0; /* get the drawing window position */
   XtSetArg(args[n],XmNx,&x);n++;
   XtSetArg(args[n],XmNy,&y);n++;
   XtGetValues(draw_para[thisone].window,args,n);

   n=0; /* set the box accordingly */
   XtSetArg(args[n],XmNx,x);n++;
   XtSetArg(args[n],XmNy,y);n++;
   XtSetValues(w,args,n);
} /* end of save_box_map_cb */


/*
save_xwdfile() obtains the id of the specified window to be dumped. It executes the X
command xwd to dump the window image into a file specified by the user. The saved file can
be viewed through using xwud.
*/

void save_xwdfile(w)
Widget w;
{
  char command[80];
  Widget shell = XtParent(draw_para[thisone].window); /* get id */

  XmUpdateDisplay(shell);
  if(XtIsManaged(w))
    { /* error check */
    printf("\nsave_box is not unmanaged yet\n");
    return;
    }
  /* set up a shell script for executing xwd */
  sprintf(command,"xwd %s -id 0x%x > %s",frame, XtWindow(shell), filename);
  system(command);
  XtFree(filename);
  filename=NULL;
} /* end of save_xwdfile() */


/*
create_device_menu() creates a device menu within a drawing window.
*/
```

```
void create_device_menu(command,canvas,whichone)
Widget command,canvas;
int whichone;
{
  Widget bt,form;
  Block_Device *blk_dev;
  int cmd_num,x,n,i;
  Arg args[10];
  XmString str;

  x = sa.index -1;
  blk_dev = sa.blk_dev;
  cmd_num = blk_dev[x].dev_no;

  /* get form widget from the drawing window */
  form = get_form_from_canvas(canvas);
  for(i=0;i<cmd_num;i++)
        { /* create button widgets and register callbacks */
        n=0;
        XtSetArg(args[n],XmNuserData,i); n++;
        device_bt[i] = XtCreateManagedWidget
          (blk_dev[x].data[i].device,
          xmPushButtonWidgetClass, command,args,n);
        XtAddCallback(device_bt[i],XmNactivateCallback,
          display_working_cursor,form);
        XtAddCallback(device_bt[i],XmNactivateCallback,draw_blk_dev_cb,canvas);
        }
  device_bt_num = i; /* save device button number */
  Device_Menu_Command = command;
} /* end of create_device_menu() */


/*
get_form_from_canvas() gets the right ancester widget and returns a pointer to it.
*/

Widget get_form_from_canvas(w)
Widget w;
{
   int i;
   for(i=0;i<3;i++)
        w = XtParent(w); /* get the right ancester widget */
   return w;
} /* end of get_form_from_canvas() */


/*
create_legend() creates for a drawing window a legend, which includes two kinds of
widgets, label and pixmap label.
*/

create_legend(parent,start_p,title,pattern,fg,bg)
Widget parent;
int start_p;
char *title,*pattern;
Pixel fg,bg;
{
  Widget label;
  int n;
  Arg args[10];
  Pixmap pixmap;

  /* create a particular pixmap pattern */
  pixmap = XmGetPixmap(XtScreen(parent), pattern,fg,bg);

  n=0; /* create a pixmap label and set its position */
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION);n++;
  XtSetArg(args[n],XmNleftPosition,start_p); n++;
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_POSITION);n++;
  XtSetArg(args[n],XmNrightPosition,start_p+1); n++;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNlabelType, XmPIXMAP); n++;
  XtSetArg(args[n],XmNlabelPixmap, pixmap); n++;
  label = XtCreateManagedWidget("label",xmLabelWidgetClass,parent,args,n);

   n=0; /* create a label associated with the pixmap label */
```

```
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION);n++;
  XtSetArg(args[n],XmNleftPosition,start_p+1); n++;
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_POSITION);n++;
  XtSetArg(args[n],XmNrightPosition,start_p+6); n++;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNlabelString,MakeXmString(title));n++;
  label = XtCreateManagedWidget("label1",
    xmLabelWidgetClass,parent,args,n);
} /* end of create_legend() */


/*
xs_register_pattern() registers a particular pattern
with the X server.
*/

xs_register_pattern(w,name,bits,width,height)
Widget w;
char *name;
unsigned char *bits;
int width,height;
{
  XImage *image;

  image = XCreateImage(XtDisplay(w),DefaultVisualOfScreen(XtScreen(w)),
    1,XYBitmap,0, bits, width, height,8,2);
  XmInstallImage(image,name);
} /* end of xs_register_pattern() */


/*
clean_pixmap() cleans the pixmap with white color */
*/

void clean_pixmap(canvas,i)
Widget canvas;
int i;
{
  Display *dpy = XtDisplay(canvas);
  int scr = DefaultScreen(dpy);

  /* clean pixmap with white */
  XSetForeground(XtDisplay(canvas),draw_para[i].gc,
    WhitePixelOfScreen(XtScreen(canvas)));
  XFillRectangle(XtDisplay(canvas),draw_para[i].pixmap,
    draw_para[i].gc,0,0,draw_para[i].width,
    draw_para[i].height);
  /* drawing is now done using black; change the gc */
  XSetForeground(XtDisplay(canvas),draw_para[i].gc,
    BlackPixelOfScreen(XtScreen(canvas)));
} /* end of clean_pixmap() */


#define BUFSIZE 900 /* buffer size for description text */

/*
help_cb() obtains a particular help text from a help file, converts it into XmString and
load  it into the help window.
*/

void help_cb(w,whichone,cbs)
Widget w;
int whichone;
XmAnyCallbackStruct *cbs;
{
  Arg    args[10];
   int n,i; char buf[BUFSIZE];
  XmString str;

  get_help_text(whichone,buf); /* get specific help text */
  str = MakeXmString(buf); /* convert text to XmString */

  if(XtIsManaged(help_box))
    XtUnmanageChild(help_box); /* unmanage first */
  n=0; /* load help message */
  XtSetArg(args[n],XmNmessageString,str); n++;
```

```
  XtSetValues(help_box,args,n);
  XtManageChild(help_box);
} /* end of help_cb() */


get_help_text() reads in a particular help message from the help file according to the
message index passed to it.
*/

get_help_text(whichone,text)
int whichone;
char *text;
{
  FILE *fp;
  char file[20],str[81],option[10];
  int length;

  sprintf(option,"#%d",whichone);
  if(whichone >9)
    length = 3;
  else
    length = 2;
  if( (fp = fopen("help.text","r")) == NULL)
    { printf("\nCannot open %s\n","help.text"); exit(1); }
  while(fgets(str,80,fp) != NULL)
    if(strncmp(str,option,length) ==0)
    break; /* find the right paragraph */
  strcpy(text,"");
  while(fgets(str,80,fp) != NULL)
    { /* read until no more */
    if(strncmp(str,option,length) ==0)
    break; /* stop at the paragraph end */
    strcat(text,str);
    }
  fclose(fp);
} /* end of get_help_text() */


/*
clear_it() clears the drawing window and set free the corrsponding pixmap.
*/

void clear_it(w,canvas,cbs)
Widget w;
Widget canvas;
XmAnyCallbackStruct *cbs;
{
  Display *dpy = XtDisplay(canvas);
  int n,i,scr = DefaultScreen(dpy);
  GC gc;
  Pixmap pixmap;
  Dimension width,height;
  Arg    args[10];

  n=0;  /* get the right pixmap to be deleted */
  XtSetArg(args[n],XmNuserData,&i); n++;
  XtGetValues(canvas,args,n);
  /* clear pixmap with white */
  XSetForeground(XtDisplay(canvas),draw_para[i].gc,
    WhitePixelOfScreen(XtScreen(canvas)));

  XFillRectangle(XtDisplay(canvas),
    draw_para[i].pixmap,  draw_para[i].gc,0,0,
    draw_para].width,draw_para[i].height);
  /* drawing is now done using black; change the gc */
  XSetForeground(XtDisplay(canvas),draw_para[i].gc,
    BlackPixelOfScreen(XtScreen(canvas)));

  XCopyArea(cbs->event->xbutton.display,
    draw_para[i].pixmap,XtWindow(canvas),
    draw_para[i].gc, 0,0, draw_para[i].width, draw_para[i].height,0,0);
  XFreePixmap(cbs->event->xbutton.display,
    draw_para[i].pixmap);
  draw_para[i].pixmap = NULL;
}  /* end of clear_it() */
```

```
/*
redraw() copies the pixmap onto its corresponding window if it exits, as a callback
procedure.
*/

void redraw(w,whichone,cbs)
Widget w;
int whichone;
XmDrawingAreaCallbackStruct *cbs;
{
  GC gc;
  Dimension width,height;

  if(draw_para[whichone].pixmap == NULL)
   return; /* no such pixmap exits */
  width = draw_para[whichone].width;
  height = draw_para[whichone].height;
  gc = draw_para[whichone].gc;
  /* copy the pixmap on to the corresponding window */
  XCopyArea(cbs->event->xexpose.display,
    draw_para[whichone].pixmap,cbs->window,gc, 0,0,width,height,0,0);
} /* end of redraw() */


/*
draw_cpu_cb(),as a callback procedure, draws pie or bar charts reflecting CPU utilization
onto a pixmap and then copies the pixmap onto the corresponding drawing window. There are
three bar charts. One shows four items as a whole; one usr% and sys%; the other wio and
idle%. The data for drawing comes from the Sa structure associated with CPU utilization
(sa.cpu_u).
*/

void draw_cpu_cb(w,canvas,cbs)  /* for cpu utilization */
Widget w;
Widget canvas;
XmAnyCallbackStruct *cbs;
{
  Display  *display = XtDisplay(canvas);
  Window   win = XtWindow(canvas);
  Widget   form;
  XGCValues values;
  GC  gc;
  Pixmap tiles[9],pixmap;
  Pixel fg,bg; Arg    args[10];
  int n,mask,i,j,x,id;
  int pos_x,pos_y,pie_size,pie_arc_s,pie_arc;
  int x_offset,y_offset,barlength,legend_xpos;
  CPU_Utilization *cpu_u;
  char legend[80];
  long pixels,mm,length;

  if(draw_para[CPU_U].pixmap) /* pixmap created */
    { /* clean the previous pixmap */
    clean_pixmap(canvas,CPU_U);
    pixmap = draw_para[CPU_U].pixmap;
    }
  else /* not yet created */
    pixmap = create_pixmap(canvas,CPU_U); /* create one */

  /* set up drawing environment */
  create_draw_env(canvas,tiles,&gc);
  XSetArcMode(display,gc,ArcPieSlice);

  n=0;
  XtSetArg(args[n],XmNuserData,&id); n++;
  XtGetValues(w,args,n);

  cpu_u = sa.cpu_u;
  if( id == 0)
    { /* draw pie */
    /* set up start position and offsets */
    pos_x = 100;  pos_y = 90; x_offset=85; y_offset=70;
    pie_size = 100;  legend_xpos = 80;
    for(i=1;i<sa.index;i++)
      {
      /* draw time title for a pie */
      XDrawImageString(display,pixmap,gc,pos_x+18,pos_y-15,
```

```
      (cpu_u+i)->time,strlen((cpu_u+i)->time));
    /* calculate the pie size for usr% */
    pie_arc = (atoi((cpu_u+i)->usr)*360)/100 * 64;
    make_legend(legend,sa.cpu_legend[1],(cpu_u+i)->usr);
    if(pie_arc >0) /* cpu% > 0 */
      { /* draw the corresponding pie section */
      /* set a particular pattern */
      XSetTile(display,gc,tiles[0]);
      XFillArc(display,pixmap,gc,pos_x,pos_y,pie_size,pie_size,0,pie_arc);
      }
    /* drawing legend */
    XDrawImageString(display,pixmap,gc,
      pos_x-legend_xpos,pos_y+25,legend,strlen(legend));
    pie_arc_s = pie_arc; /* save the previous arc size */
    /* calculate the pie arc value for sys% */
    pie_arc = ( atoi((cpu_u+i)->sys)*360)/100 * 64;
    if(pie_arc >0) /* % > 0 */
      { /* draw pie section */
      XSetTile(display,gc,tiles[4]);
      XFillArc(display,pixmap,gc,pos_x,pos_y,pie_size,pie_size,pie_arc_s,pie_arc);
      }
    make_legend(legend,sa.cpu_legend[2],(cpu_u+i)->sys);
    XDrawImageString(display,pixmap,gc,pos_x-legend_xpos,pos_y+40,
      legend,strlen(legend) );
    pie_arc_s += pie_arc; /* accumulate the arc value */
    pie_arc = (atoi((cpu_u+i)->wio)*360)/100 * 64;
    make_legend(legend,sa.cpu_legend[3],(cpu_u+i)->wio);
    if(pie_arc >0)
      {
      /* make up for the difference from 100% */
      if( atoi((cpu_u+i)->idle) == 0)
            pie_arc = 360*64 - pie_arc_s;
      XSetTile(display,gc,tiles[2]);
      XFillArc (display,pixmap,gc,pos_x,pos_y,pie_size,pie_size,pie_arc_s,pie_arc);
      }
    XDrawImageString(display,pixmap,gc,pos_x-legend_xpos,pos_y+55,
          legend,strlen(legend) );

    pie_arc_s += pie_arc; /* same approach as above */
    pie_arc = ( atoi((cpu_u+i)->idle)*360)/100 * 64;
    make_legend(legend,sa.cpu_legend[4],(cpu_u+i)->idle);
    if(pie_arc >0)
      {
      pie_arc = 360*64 - pie_arc_s;
      XSetTile(display,gc,tiles[8]);
      XFillArc (display,pixmap,gc,pos_x,pos_y,
          pie_size,pie_size,pie_arc_s,pie_arc);
      }
    XDrawImageString(display,pixmap,gc,pos_x-legend_xpos,
      pos_y+70,legend,strlen(legend));

    XSetTile(display,gc,tiles[0]);
    XDrawArc (display,pixmap,gc,pos_x,pos_y,pie_size,pie_size,0,360*64);
    /* update x position */
    pos_x += pie_size + x_offset + 20; /* change y position when four pies in row */
    if(i%4 == 0)
      { pos_x = 100; pos_y += 160; }
    } /* for */
  } /* end if id == 0 */

else if ( id == 1)
  {
  /* draw bar chart */
  length = get_display_length(display);
  pos_x = X_START; pos_y = 70;
  x_offset = 2*length/10; y_offset = 15;
  XSetTile(display,gc,tiles[0]);
  /* draw scale lines */
  draw_rectangle(display,pixmap,gc,
      pos_x,pos_y,2*length,sa.index*15);
  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
      tiles[4],tiles[0],"50","100","",0,0);
  draw_time(display,pixmap,gc,pos_x-90,
      pos_y,y_offset,sa.index);

  for(i=1;i<sa.index;i++)
     { XSetTile(display,gc,tiles[0]); /* specify pattern */
```

```
        /* calculate bar length for usr% */
        barlength = 2*length*(atoi((cpu_u+i)->usr))/100;
        /* draw usr% bar */
        XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
        XSetTile(display,gc,tiles[4]);
        /* draw sys% bar */
        XFillRectangle (display,pixmap,gc,pos_x+barlength,pos_y+(i-1)*y_offset,
          2*length*(atoi((cpu_u+i)->sys))/100,10);
        barlength += 2*length*(atoi((cpu_u+i)->sys))/100;
        XSetTile(display,gc,tiles[2]);
        /* draw wio% bar */
        XFillRectangle (display,pixmap,gc,pos_x+barlength,pos_y+(i-1)*y_offset,
          2*length*(atoi((cpu_u+i)->wio))/100,10);
        barlength += 2*length*(atoi((cpu_u+i)->wio))/100;
        XSetTile(display,gc,tiles[8]);
        /* draw idle% bar */
        XFillRectangle (display,pixmap,gc,pos_x+barlength,
          pos_y+(i-1)*y_offset,2*length-barlength,10);
        } /* for */

    /* x and y positions for next chart */
    pos_y += i*y_offset+80;
    /* draw %usr and %sys with same approach as above */
    XSetTile(display,gc,tiles[0]);
    draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
    draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
      tiles[4],tiles[0],"100","0","%usr and %sys",-5,30);
    draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);

    XSetTile(display,gc,tiles[0]);
    for(i=1;i<sa.index;i++)
         { /* draw usr% bar */
         barlength = length*(atoi((cpu_u+i)->usr))/100;
         XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
         }

    pos_x += 2*length;
    draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
    XSetTile(display,gc,tiles[4]);
    for(i=1;i<sa.index;i++)
         { /* draw sys% bar */
         barlength = length*(atoi((cpu_u+i)->sys))/100;
         XFillRectangle (display,pixmap,gc,pos_x-barlength,
           pos_y+(i-1)*y_offset,barlength,10);
         }

    /* x and y positions for next chart */
    pos_y += i*y_offset+80;  pos_x -= 2*length;
      /* draw %wio and %idle with same approach as above */
    XSetTile(display,gc,tiles[0]);
    draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
    draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
        tiles[4],tiles[0],"100","0","%wio and %idle",-5,14);
    draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);

    XSetTile(display,gc,tiles[2]);
    for(i=1;i<sa.index;i++)
         { /* draw wio% bar */
         barlength = length*(atoi((cpu_u+i)->wio))/100;
         XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
         }

    pos_x += 2*length;
    draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
    XSetTile(display,gc,tiles[8]);
    for(i=1;i<sa.index;i++)
         { /* draw idle% bar */
         barlength = length*(atoi((cpu_u+i)->idle))/100;
         XFillRectangle (display,pixmap,gc,pos_x-barlength,
           pos_y+(i-1)*y_offset,barlength,10);
         }
    } /* else if */
else { printf("\nreturned id = %d\n",id); exit(1); }
/* copy pixmap onto the corresponding drawing window */
XCopyArea(display,pixmap,win,draw_para[CPU_U].gc,0,0, draw_para[CPU_U].width,
   draw_para[CPU_U].height, 0,0);
draw_para[CPU_U].pixmap = pixmap; /* store pixmap id */
```

```
  form = canvas;
  for(i=0;i<3;i++) /* find the ancester form */
      form = XtParent(form);
  /* change busy cursor to default one */
  XUndefineCursor(display,XtWindow(form));
} /* end of draw_cpu_cb() */


/*
create_gc() creates graphic content for the drawing window.
It returns a pointer to the created gc.
*/

GC create_gc(canvas)
Widget canvas;
{
  int mask,n;
  XGCValues values;
  GC gc;
  Arg args[10];

  mask = GCForeground | GCBackground | GCFillStyle;
  /* get the colors used by the widget */
   n=0; XtSetArg(args[n],XmNforeground, &values.foreground); n++;
  XtSetArg(args[n],XmNbackground, &values.background); n++;
  XtGetValues(canvas,args,n);

  values.fill_style = FillTiled; /* set file style */
  gc = XtGetGC(canvas,mask,&values); /* create gc */
  return gc;
} /* end of create_gc() */


/*
create_draw_env() creates an environment for pixmap drawing,
which includes tile setting and gc creation.
*/

void create_draw_env(canvas,tiles,gc)
Widget canvas;
Pixmap *tiles;
GC *gc;
{
  Pixel fg,bg;
  Arg args[10];
  int n,i;

  /* get the colors used by the widget */
  n=0;  XtSetArg(args[n],XmNforeground, &fg); n++;
  XtSetArg(args[n],XmNbackground, &bg); n++;
  XtGetValues(canvas,args,n);

  for(i=0;i<XtNumber(patterns);i++) /* set tiles */
    tiles[i] = XmGetPixmap(XtScreen(canvas), patterns[i],fg,bg);
  *gc = create_gc(canvas);
}  /* end of create_draw_env() */


/*
create_help_box() creates a message box of motif-specified InformationDialog class and
returns a pointer to the box.
The box is used to display help message.
*/

Widget create_help_box(bt)
Widget bt;
{
  XmString str;
  Position x,y;
  Widget dia_box;
  Arg args[10];
  int n;

  x = 627; y=84; /* position of the help box */
  n=0;
  XtSetArg(args[n],XmNdefaultPosition,False); n++;
```

```
  XtSetArg(args[n],XmNx,x);n++;
  XtSetArg(args[n],XmNy,y);n++;
  dia_box = XmCreateInformationDialog(bt,"help",args,n);
  /* remove the cancel and help buttons */
  XtUnmanageChild(XmMessageBoxGetChild(dia_box,XmDIALOG_CANCEL_BUTTON));
  XtUnmanageChild(XmMessageBoxGetChild(dia_box,XmDIALOG_HELP_BUTTON));
  return (dia_box);
} /* end of create_help_box() */


/*
draw_buf_cb(), as a callback procedure, draws bar charts reflecting buffer utilization. It
uses the same approach as the one in draw_cpu_cb(), and thus similar in-line comments will
be omitted.
*/

void draw_buf_cb(w,canvas,cbs)
Widget w; Widget canvas;
XmAnyCallbackStruct *cbs;
{
  Display  *display = XtDisplay(canvas);
  Window   win = XtWindow(canvas);
  XGCValues values;  GC         gc;
  Pixmap tiles[9],pixmap;
  Arg    args[10];
  int n,i,j,x,pos_x,pos_y,x_offset,y_offset,barlength;
  Buffer_Activity *buf;
  Y_axis *y_axis;
  long length;
  int max_int,max_int1,max_int2,scaler_int;
  char mid_str[10],max_str[10],legend[80];

  if(draw_para[BUF].pixmap) /* pixmap existing */
    return;
  pixmap = create_pixmap(canvas,BUF); /* create one */

  create_draw_env(canvas,tiles,&gc);
  buf = sa.buf;
  y_axis = sa.y_axis;
  length = get_display_length(display);

  pos_x = X_START; pos_y = 70; /* set start position */
  XSetTile(display,gc,tiles[0]);
  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  x_offset = 2*length/10;  y_offset = 15; /* set offset */
  /* determin the maximum value for drawing */
  max_int = 100;
  strcpy(max_str,"100");
  strcpy(mid_str,"");
  scaler_int = max_int;
  max_int1 = (buf+1)->bread;
  for(i=2;i<sa.index;i++) /* get maximum value for bread */
    if((buf+i)->bread > max_int1)
      max_int1 = (buf+i)->bread;
  max_int2 = (buf+1)->bwrite;
  for(i=2;i<sa.index;i++) /* get maximum value for bwrited */
    if((buf+i)->bwrite > max_int2)
      max_int2 = (buf+i)->bwrite;
  if(max_int2 > max_int1)
    max_int1 = max_int2;
  if(max_int1 > max_int)
    max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
    tiles[4],tiles[0],max_str,"0","bread/s vs. bwrite/s",- 5,0);
  draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[0]);
  for(i=1;i<sa.index;i++)
    { /* draw bread chart */
    barlength = length*((buf+i)->bread)/max_int;
    XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
    }

  pos_x += 2*length; /* update x position */

  draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[3]);
```

```
for(i=1;i<sa.index;i++)
  { /* draw bwrite chart */
  barlength = length*((buf+i)->bwrite)/max_int;
  XFillRectangle(display,pixmap,gc,pos_x-barlength,pos_y+(i-1)*y_offset,barlength,10);
  }
/* x and y positions of the next chart */
pos_y += i*y_offset+80;
pos_x -= 2*length;

/* draw lread and lwrite */
max_int = 500;
strcpy(max_str,"500");
strcpy(mid_str,"");
scaler_int = max_int;
max_int1 = (buf+1)->lread;
for(i=2;i<sa.index;i++)
  if((buf+i)->lread > max_int1)
    max_int1 = (buf+i)->lread;
max_int2 = (buf+1)->lwrite;
for(i=2;i<sa.index;i++)
  if((buf+i)->lwrite > max_int2)
    max_int2 = (buf+i)->lwrite;
if(max_int2 > max_int1)
  max_int1 = max_int2;
if(max_int1 > max_int)
  max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

XSetTile(display,gc,tiles[0]);
draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
  tiles[4],tiles[0],max_str,"0","lread/s vs.  lwrite/s",- 5,0);
draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset, sa.index);

XSetTile(display,gc,tiles[1]);
for(i=1;i<sa.index;i++)
  { /* draw lread chart */
  barlength = length*((buf+i)->lread)/max_int;
  XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
  }
pos_x += 2*length;
draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index) ;

XSetTile(display,gc,tiles[4]);
for(i=1;i<sa.index;i++)
  { /* draw lwrite chart */
  barlength = length*((buf+i)->lwrite)/max_int;
  XFillRectangle(display,pixmap,gc,pos_x-barlength,pos_y+(i-1)*y_offset,barlength,10);
  }
/* draw %rcache and %wcache */
pos_y += i*y_offset+80;
pos_x -= 2*length;
XSetTile(display,gc,tiles[0]);
draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
  tiles[4],tiles[0],"100","0","%rcache vs.  %wcache",-5,0);
draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);

XSetTile(display,gc,tiles[2]);
for(i=1;i<sa.index;i++)
  { /* draw rcache chart */
  barlength = length*((buf+i)->rcache)/100;
  XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
  }
pos_x += 2*length;
draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);

XSetTile(display,gc,tiles[5]);
for(i=1;i<sa.index;i++)
  { /* draw wcache chart */
  barlength = length*((buf+i)->wcache)/100;
  XFillRectangle(display,pixmap,gc,pos_x-barlength,pos_y+(i-1)*y_offset,barlength,10);
  }
/* draw pread/s and pwrit/s */
pos_y += i*y_offset+80;
pos_x -= 2*length;

max_int = 100;
```

```
  strcpy(max_str,"100");
  strcpy(mid_str,"");
  scaler_int = max_int;
  max_int1 = (buf+1)->pread;
  for(i=2;i<sa.index;i++)
    if((buf+i)->pread > max_int1)
      max_int1 = (buf+i)->pread;
  max_int2 = (buf+1)->pwrite;
  for(i=2;i<sa.index;i++)
    if((buf+i)->pwrite > max_int2)
      max_int2 = (buf+i)->pwrite;
  if(max_int2 > max_int1)
    max_int1 = max_int2;
  if(max_int1 > max_int)
    max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

  XSetTile(display,gc,tiles[0]);
  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
    tiles[4],tiles[0],max_str,"0","pread/s vs.  pwrite/s",- 5,0);
  draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index) ;

  XSetTile(display,gc,tiles[6]);
  for(i=1;i<sa.index;i++)
    { /* draw pread chart */
    barlength = length*((buf+i)->pread)/max_int;
    XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
    }

  pos_x += 2*length;
  draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[7]);
  for(i=1;i<sa.index;i++)
    { /* draw pwrite chart */
    barlength = length*((buf+i)->pwrite)/max_int;
    XFillRectangle (display,pixmap,gc,pos_x-barlength,pos_y+(i-1)*y_offset,barlength,10);
    }

  XCopyArea(display,pixmap,win,draw_para[BUF].gc,
    0,0, draw_para[BUF].width, draw_para[BUF].height, 0,0);
  draw_para[BUF].pixmap = pixmap;
  /* set cousor back to default */
  XUndefineCursor(display,XtWindow(graph_box_globe));
} /* end of draw_buf_cb() */


/*
draw_blk_dev_cb, as a callback procedure, draws bar charts reflecting block device
activity. It uses the same approach as the one in draw_cpu_cb(), and thus similar in-line
comments will be omitted.
*/

void draw_blk_dev_cb(w,canvas,cbs)
Widget w;
Widget canvas;
XmAnyCallbackStruct *cbs;
{
  Display  *display = XtDisplay(canvas);
  Window   win = XtWindow(canvas);
  Widget form;
  XGCValues values;
  GC      gc;
  Pixmap tiles[9],pixmap;
  Arg    args[10];
  int n,i,id,j,x;
  int pos_x,pos_y,x_offset,y_offset,barlength;
  Block_Device *blk_dev;
  Y_axis *y_axis;
  char legend[80],dev_name[80],mid_str[10],max_str[10];
  long length;
  int max_int,max_int1,max_int2,scaler_int;

  if(draw_para[BLK_DEV].pixmap) /* pixmap created */
    { /* clean the pixmap */
    clean_pixmap(canvas,BLK_DEV);
    pixmap = draw_para[BLK_DEV].pixmap;
    }
```

```
else /* create one */
  pixmap = create_pixmap(canvas,BLK_DEV);

create_draw_env(canvas,tiles,&gc);
blk_dev = sa.blk_dev;
y_axis = sa.y_axis;

n=0;
XtSetArg(args[n],XmNuserData,&id); n++;
XtGetValues(w,args,n);
length = get_display_length(display);
/* set position */
pos_x = X_START; pos_y = 70;
XSetTile(display,gc,tiles[0]);
draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
x_offset = 2*length/10;
y_offset = 15;
/* determin the maximum value for drawing */
max_int = 100;
strcpy(max_str,"100");
strcpy(mid_str,"");
scaler_int = max_int;
max_int1 = (blk_dev+1)->data[id].avque;
for(i=2;i<sa.index;i++)
  if((blk_dev+i)->data[id].avque > max_int1)
    max_int1 = (blk_dev+i)->data[id].avque;
if(max_int1 > max_int)
  max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);
sprintf(dev_name,"%busy and avque(%s)",(blk_dev+(sa.index-1))->data[id].device);
draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
  tiles[4],tiles[0],max_str,"0",dev_name,-5,0);

draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);

XSetTile(display,gc,tiles[0]);
for(i=1;i<sa.index;i++)
  { /* draw busy chart */
  if( id > (blk_dev+i)->dev_no-1)
    continue;
  barlength = length*((blk_dev+i)->data[id].busy)/max_int;
  XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
  }

pos_x += 2*length;  /* up x position */

draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index) ;

XSetTile(display,gc,tiles[1]);
for(i=1;i<sa.index;i++)
  { /* draw average queue length */
  if( id > (blk_dev+i)->dev_no-1)
    continue;
  barlength = length*((blk_dev+i)->data[id].avque)/max_int;
  XFillRectangle (display,pixmap,gc,pos_x-barlength,pos_y+(i-1)*y_offset,barlength,10);
  }
/* x and y positions of the next chart */
pos_y += i*y_offset+80;
pos_x -= 2*length;

/* get maximum value for drawing */
max_int = 500;
strcpy(max_str,"500");
strcpy(mid_str,"");
scaler_int = max_int;
max_int1 = (blk_dev+1)->data[id].r_w;
for(i=2;i<sa.index;i++)
 if((blk_dev+i)->data[id].r_w > max_int1)
      max_int1 = (blk_dev+i)->data[id].r_w;
max_int2 = (blk_dev+1)->data[id].blks;
for(i=2;i<sa.index;i++)
 if((blk_dev+i)->data[id].blks > max_int2)
      max_int2 = (blk_dev+i)->data[id].blks;
if(max_int2 > max_int1)
 max_int1 = max_int2;
if(max_int1 > max_int)
   max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);
```

```
sprintf(dev_name,"r+w/s and blks/s(%s)",(blk_dev+1)- >data[id].device);
XSetTile(display,gc,tiles[0]);
draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
  tiles[4],tiles[0],max_str,"0",dev_name,-5,0);
draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);

XSetTile(display,gc,tiles[2]);
for(i=1;i<sa.index;i++)
  {
  if( id > (blk_dev+i)->dev_no-1)
    continue;
  barlength = length*((blk_dev+i)->data[id].r_w)/max_int;
  XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
  }
pos_x += 2*length; /* update x position */

draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);

XSetTile(display,gc,tiles[3]); /* set file pattern */
for(i=1;i<sa.index;i++)
  {  /* draw block activity */
  if( id > (blk_dev+i)->dev_no-1)
    continue;
  barlength = length*((blk_dev+i)->data[id].blks)/max_int;
  XFillRectangle(display,pixmap,gc,pos_x-barlength,pos_y+(i-1)*y_offset,barlength,10);
  }
/* draw rcache and wcache */
pos_y += i*y_offset+80;
pos_x -= 2*length;

max_int = 100;
strcpy(max_str,"100");
strcpy(mid_str,"");
scaler_int = max_int;
max_int1 = (blk_dev+1)->data[id].avwait;
for(i=2;i<sa.index;i++)
    if((blk_dev+i)->data[id].avwait > max_int1)
      max_int1 = (blk_dev+i)->data[id].avwait;
max_int2 = (blk_dev+1)->data[id].avserv;
for(i=2;i<sa.index;i++)
    if((blk_dev+i)->data[id].avserv > max_int2)
      max_int2 = (blk_dev+i)->data[id].avserv;
if(max_int2 > max_int1)
    max_int1 = max_int2;
if(max_int1 > max_int)
    max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

sprintf(dev_name,"avwait vs. avserv (%s)",(blk_dev+1)->data[id].device);
XSetTile(display,gc,tiles[0]);

draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
    tiles[4],tiles[0],max_str,"0",dev_name,-5,0);
draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);

XSetTile(display,gc,tiles[4]);
for(i=1;i<sa.index;i++)
    {/* draw avwait chart */
    if( id > (blk_dev+i)->dev_no-1)
      continue;
    barlength = length*((blk_dev+i)->data[id].avwait)/max_int ;
    XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
    }

pos_x += 2*length;

draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
 XSetTile(display,gc,tiles[5]); for(i=1;i<sa.index;i++)
  { /* draw avserv chart */
  if( id > (blk_dev+i)->dev_no-1)
    continue;
  barlength = length*((blk_dev+i)->data[id].avserv)/max_int ;
  XFillRectangle(display,pixmap,gc,pos_x-barlength,pos_y+(i-1)*y_offset,barlength,10);
  }

XCopyArea(display,pixmap,win,draw_para[BLK_DEV].gc,
```

```
      0,0, draw_para[BLK_DEV].width, draw_para[BLK_DEV].height, 0,0);
  draw_para[BLK_DEV].pixmap = pixmap;
  form = canvas;
  for(i=0;i<3;i++)
      form = XtParent(form);
  XUndefineCursor(display,XtWindow(form));
}  /* end of draw_blk_dev_cb */


/*
draw_tty_cb(), as a callback procedure, draws bar charts reflecting TTY activity. It uses
the same approach as the one in draw_cpu_cb(), and thus similar in-line comments will be
omitted.
*/

void draw_tty_cb(w,canvas,cbs)Widget w;
Widget canvas;
XmAnyCallbackStruct *cbs;
{
  Display  *display = XtDisplay(canvas);
  Window   win = XtWindow(canvas);
  XGCValues values;
  GC      gc;
  Pixmap tiles[9],pixmap;
  Pixel fg,bg;
  Arg    args[10];
  int n,mask,i,j,x,id;
  int pos_x,pos_y,pie_size,pie_arc_s,pie_arc;
  int x_offset,y_offset,barlength;
  TTY *tty;
  Y_axis *y_axis;
  char legend[80],mid_str[10],max_str[10];
  long pixels,mm,length;
  int myscreen;
  int max_int,max_int1,max_int2,scaler_int;

  if(draw_para[TTY_].pixmap)
    return;
  pixmap = create_pixmap(canvas,TTY_);

  create_draw_env(canvas,tiles,&gc);
  tty = sa.tty;
  y_axis = sa.y_axis;
  length = get_display_length(display);

  /* set up start position */
  pos_x = X_START; pos_y = 70;
  XSetTile(display,gc,tiles[0]);

  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  x_offset = 2*length/10;
  y_offset = 15;

  /* determin the maximum value for drawing */
  max_int = 500;
  strcpy(max_str,"500");
  strcpy(mid_str,"");
  scaler_int = max_int;
  max_int1 = (tty+1)->rawch;
  for(i=2;i<sa.index;i++)
    if((tty+i)->rawch > max_int1)
      max_int1 = (tty+i)->rawch;
  max_int2 = (tty+1)->canch;
  for(i=2;i<sa.index;i++)
    if((tty+i)->canch > max_int2)
      max_int2 = (tty+i)->canch;
  if(max_int2 > max_int1)
    max_int1 = max_int2;
  if(max_int1 > max_int)
    max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
    tiles[4],tiles[0],max_str,"0","rawch/s and canch/s",-5,0) ;
  draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);

  XSetTile(display,gc,tiles[0]);
    for(i=1;i<sa.index;i++)
```

```
   {  /* draw rawch chart */
   barlength = length*((tty+i)->rawch)/max_int;
   XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
   }

pos_x += 2*length; /* update x */
draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
XSetTile(display,gc,tiles[1]);
for(i=1;i<sa.index;i++)
    { /* draw canch chart */
    barlength = length*((tty+i)->canch)/max_int;
    XFillRectangle (display,pixmap,gc,pos_x-barlength,
      pos_y+(i-1)*y_offset,barlength,10);
    }
/* x and y positions of next chart */
pos_y += i*y_offset+80;
pos_x -= 2*length;
/* determin the maximum value for drawing */
max_int = 500;
strcpy(max_str,"500");
strcpy(mid_str,"");
scaler_int = max_int;
max_int1 = (tty+1)->outch;
for(i=2;i<sa.index;i++)
    if((tty+i)->outch > max_int1)
      max_int1 = (tty+i)->outch;
max_int2 = (tty+1)->rcvin;
for(i=2;i<sa.index;i++)
    if((tty+i)->rcvin > max_int2)
      max_int2 = (tty+i)->rcvin;
if(max_int2 > max_int1)
    max_int1 = max_int2;
if(max_int1 > max_int)
    max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

XSetTile(display,gc,tiles[0]);
draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
  tiles[4],tiles[0],max_str,"0","outch/s and rcvin/s",-5,0) ;
draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index) ;
XSetTile(display,gc,tiles[2]);
    for(i=1;i<sa.index;i++)
      { /* draw outch chart */
      barlength = length*((tty+i)->outch)/max_int;
      XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
      }

pos_x += 2*length;
draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index) ;
XSetTile(display,gc,tiles[3]);
for(i=1;i<sa.index;i++)
    { /* draw rcvin chart */
    barlength = length*((tty+i)->rcvin)/max_int;
    XFillRectangle (display,pixmap,gc,pos_x-barlength,
      pos_y+(i-1)*y_offset,barlength,10);
    }

pos_y += i*y_offset+80;
pos_x -= 2*length;
max_int = 500;
strcpy(max_str,"500");
strcpy(mid_str,"");
scaler_int = max_int;
max_int1 = (tty+1)->xmtin;
for(i=2;i<sa.index;i++)
    if((tty+i)->xmtin > max_int1)
      max_int1 = (tty+i)->xmtin;
max_int2 = (tty+1)->mdmin;
for(i=2;i<sa.index;i++)
    if((tty+i)->mdmin > max_int2)
      max_int2 = (tty+i)->mdmin;
if(max_int2 > max_int1)
    max_int1 = max_int2;
if(max_int1 > max_int)
    max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

XSetTile(display,gc,tiles[0]);
```

```
  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
      tiles[4],tiles[0],max_str,"0","xmtin/s and mdmin/s",-5,0) ;
  draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index) ;
  XSetTile(display,gc,tiles[4]);
  for(i=1;i<sa.index;i++)
      { /* draw xmtin chart */
      barlength = length*((tty+i)->xmtin)/max_int;
      XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
      }

  pos_x += 2*length;
  draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[5]);
      for(i=1;i<sa.index;i++)
        {  /* draw mdmin chart */
        barlength = length*((tty+i)->mdmin)/max_int;
        XFillRectangle (display,pixmap,gc,pos_x-barlength,
          pos_y+(i-1)*y_offset,barlength,10);
        }

  XCopyArea(display,pixmap,win,draw_para[TTY_].gc,
      0,0, draw_para[TTY_].width, draw_para[TTY_].height, 0,0);
  draw_para[TTY_].pixmap = pixmap;
  XUndefineCursor(display,XtWindow(graph_box_globe));
} /* end of draw_tty_cb() */


/*
draw_sysc_cb(), as a callback procedure, draws bar charts reflecting system call activity.
It uses the same approach as the one in draw_cpu_cb(), and thus similar in-line comments
will be omitted.
*/

void draw_sysc_cb(w,canvas,cbs)
Widget w;
Widget canvas;
XmAnyCallbackStruct *cbs;
{
  Display  *display = XtDisplay(canvas);
  Window   win = XtWindow(canvas);
  XGCValues values;
  GC    gc;
  Pixmap tiles[9],pixmap;
  Arg    args[10];
  int n,i,j,x;
  int pos_x,pos_y,x_offset,y_offset,barlength;
  Sys_Call *sysc;
  Y_axis *y_axis;
  char legend[80],mid_str[10],max_str[10];
  long length;
  float max_ft1,max_ft2,max_ft,scaler_ft;
  int max_int,max_int1,max_int2,scaler_int;

  if(draw_para[SYSC].pixmap)
      return;
  pixmap = create_pixmap(canvas,SYSC);

  create_draw_env(canvas,tiles,&gc);
  sysc = sa.sysc;
  y_axis = sa.y_axis;
  length = get_display_length(display);

  /* set starting position */
  pos_x = X_START; pos_y = 70;
  XSetTile(display,gc,tiles[0]);
  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  x_offset = 2*length/10;
  y_offset = 15;
  /* determin the maximum value for drawing */
  max_int = 1000;
  strcpy(max_str,"1000");
  strcpy(mid_str,"500");
  scaler_int = max_int / 10;
  max_int1 = (sysc+1)->scall;
  for(i=2;i<sa.index;i++)
    if((sysc+i)->scall > max_int1)
```

```
    max_int1 = (sysc+i)->scall;
if(max_int1 > max_int)
  max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
  tiles[4],tiles[4],mid_str,max_str,"scall/s",-10,60);
draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
XSetTile(display,gc,tiles[0]);
for(i=1;i<sa.index;i++)
  { /* draw scall chart */
  barlength = 2*length*((sysc+i)->scall)/max_int;
  XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
  }

/* draw sread and swrite */
/* x and y positions of next chart */
pos_y += i*y_offset+80;
/* determin the maximum value for drawing */
max_int = 500;
strcpy(max_str,"500");
strcpy(mid_str,"");
scaler_int = max_int / 10;
max_int1 = (sysc+1)->sread;
for(i=2;i<sa.index;i++)
  if((sysc+i)->sread > max_int1)
    max_int1 = (sysc+i)->sread;
max_int2 = (sysc+1)->swrit;
for(i=2;i<sa.index;i++)
  if((sysc+i)->swrit > max_int2)
    max_int2 = (sysc+i)->swrit;
if(max_int2 > max_int1)
  max_int1 = max_int2;
if(max_int1 > max_int)
  max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

XSetTile(display,gc,tiles[0]);
draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
  tiles[4],tiles[0],max_str,"0","sread/s vs. swrite/s",-5,0);
draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index) ;

XSetTile(display,gc,tiles[1]);
for(i=1;i<sa.index;i++)
  { /* draw sread chart */
  barlength = length*((sysc+i)->sread)/max_int;
  XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
  }
pos_x += 2*length;

draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
XSetTile(display,gc,tiles[2]);
for(i=1;i<sa.index;i++)
  { /* draw swrit chart */
  barlength = length*((sysc+i)->swrit)/max_int;
  XFillRectangle (display,pixmap,gc,pos_x-barlength,pos_y+(i-1)*y_offset,barlength,10);
  }

/* draw fork/s and exec/s */
pos_y += i*y_offset+80;
pos_x -= 2*length;
max_int = 100;
strcpy(max_str,"100");
strcpy(mid_str,"");
scaler_int = max_int / 10;
max_int1 = (sysc+1)->fork;
for(i=2;i<sa.index;i++)
  if((sysc+i)->fork > max_int1)
    max_int1 = (sysc+i)->fork;
max_int2 = (sysc+1)->exec;
for(i=2;i<sa.index;i++)
  if((sysc+i)->exec > max_int2)
    max_int2 = (sysc+i)->exec;
if(max_int2 > max_int1)
  max_int1 = max_int2;
if(max_int1 > max_int)
  max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);
```

```
XSetTile(display,gc,tiles[0]);

draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
  tiles[4],tiles[0],max_str,"0","fork/s and exec/s",-5,5);

draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
 XSetTile(display,gc,tiles[3]); for(i=1;i<sa.index;i++)
  {  /* draw fork chart */
  barlength = length*((sysc+i)->fork)/max_int;
  XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
  }

pos_x += 2*length;
draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
XSetTile(display,gc,tiles[4]);
for(i=1;i<sa.index;i++)
  { /* draw exec chart */
  barlength = length*((sysc+i)->exec)/max_int;
  XFillRectangle (display,pixmap,gc,pos_x-barlength,pos_y+(i-1)*y_offset,barlength,10);
  }

/* draw pread/s and pwrit/s */
pos_y += i*y_offset+80;
pos_x -= 2*length;
max_int = 500000;
strcpy(max_str,"500000");
strcpy(mid_str,"");
scaler_int = max_int / 10;
max_int1 = (sysc+1)->rchar;
for(i=2;i<sa.index;i++)
  if((sysc+i)->rchar > max_int1)
    max_int1 = (sysc+i)->rchar;
max_int2 = (sysc+1)->wchar;
for(i=2;i<sa.index;i++)
  if((sysc+i)->wchar > max_int2)
    max_int2 = (sysc+i)->wchar;
if(max_int2 > max_int1)
  max_int1 = max_int2;
if(max_int1 > max_int)
  max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

XSetTile(display,gc,tiles[0]);
draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
  tiles[4],tiles[0],max_str,"0","rchar/s vs. wchar/s",-16,0);

draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
XSetTile(display,gc,tiles[5]);
for(i=1;i<sa.index;i++)
    { /* draw rchar chart */
    barlength = length*((sysc+i)->rchar)/max_int;
    if(barlength < 0 )
       { /* draw differently owing to original data error */
       barlength = absa(barlength);
       XSetTile(display,gc,tiles[7]);
       XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
       XSetTile(display,gc,tiles[5]);
       }
    else /* normal drawing */
       XFillRectangle(display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
    }

pos_x += 2*length;
draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
XSetTile(display,gc,tiles[6]);
for(i=1;i<sa.index;i++)
    { /* draw wchar chart */
    barlength = length*((sysc+i)->wchar)/max_int;
    if(barlength < 0 )
       { /* draw differently owing to original data error */
       barlength = absa(barlength);
       XSetTile(display,gc,tiles[7]);
       XFillRectangle (display,pixmap,gc,pos_x-barlength,
         pos_y+(i-1)*y_offset,barlength,10);
       XSetTile(display,gc,tiles[6]);
       }
```

```
      else /* normal drawing */
        XFillRectangle (display,pixmap,gc,pos_x-barlength,
          pos_y+(i-1)*y_offset,barlength,10);
      }

  XCopyArea(display,pixmap,win,draw_para[SYSC].gc,
      0,0, draw_para[SYSC].width,draw_para[SYSC].height, 0,0);
  draw_para[SYSC].pixmap = pixmap;
  XUndefineCursor(display,XtWindow(graph_box_globe));
} /* end of draw_sysc_cb() */


/*
draw_syss_cb(), as a callback procedure, draws bar charts reflecting system swapping
activity. It uses the same approach as the one in draw_cpu_cb(), and thus similar in- line
comments will be omitted.
*/
void draw_syss_cb(w,canvas,cbs)
Widget w;
Widget canvas;
XmAnyCallbackStruct *cbs;
{
  Display  *display = XtDisplay(canvas);
  Window   win = XtWindow(canvas);
  XGCValues values;
  GC    gc;
  Pixmap tiles[9],pixmap;
  Arg    args[10];
  int n,i,j,x;
  int pos_x,pos_y,x_offset,y_offset,barlength; Sys_Swap *syss;
  Y_axis *y_axis;
  char legend[80],mid_str[10],max_str[10];
  long length;
  int max_int,max_int1,max_int2,scaler_int;

  if(draw_para[SYSS].pixmap)
    return;
  pixmap = create_pixmap(canvas,SYSS);
  create_draw_env(canvas,tiles,&gc);
  syss = sa.syss;
  y_axis = sa.y_axis;
  length = get_display_length(display);

  /* set starting positions */
  pos_x = X_START; pos_y = 70;
  XSetTile(display,gc,tiles[0]);
  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  x_offset = 2*length/10;
  y_offset = 15;
  /* determin the maximum value for drawing */
  max_int = 100;
  strcpy(max_str,"100");
  strcpy(mid_str,"");
  scaler_int = max_int;
  max_int1 = (syss+1)->swpin;
  for(i=2;i<sa.index;i++)
    if((syss+i)->swpin > max_int1)
      max_int1 = (syss+i)->swpin;
  max_int2 = (syss+1)->swpot;
  for(i=2;i<sa.index;i++)
    if((syss+i)->swpot > max_int2)
      max_int2 = (syss+i)->swpot;
  if(max_int2 > max_int1)
    max_int1 = max_int2;
  if(max_int1 > max_int)
    max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
      tiles[4],tiles[0],max_str,"0","swpin/s vs. swpot/s",-5,0) ;
  draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[0]);
  for(i=1;i<sa.index;i++)
    { /* draw swpin chart */
    barlength = length*((syss+i)->swpin)/max_int;
    XFillRectangle (display,pixmap,gc,pos_x,
    pos_y+(i-1)*y_offset,barlength,10);
    }
```

```
pos_x += 2*length;
draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
XSetTile(display,gc,tiles[2]);
for(i=1;i<sa.index;i++)
  { /* draw swpot chart */
  barlength = length*((syss+i)->swpot)/max_int;
  XFillRectangle (display,pixmap,gc,pos_x-barlength,pos_y+(i-1)*y_offset,barlength,10);
  }
/* x and y positions of next chart */
pos_y += i*y_offset+80;
pos_x -= 2*length;
/* draw bswin and bswot */
max_int = 100;
strcpy(max_str,"100");
strcpy(mid_str,"");
scaler_int = max_int;
max_int1 = (syss+1)->bswin;
for(i=2;i<sa.index;i++)
  if((syss+i)->bswin > max_int1)
    max_int1 = (syss+i)->bswin;
max_int2 = (syss+1)->bswot;
for(i=2;i<sa.index;i++)
  if((syss+i)->bswot > max_int2)
    max_int2 = (syss+i)->bswot;
if(max_int2 > max_int1)
  max_int1 = max_int2;
if(max_int1 > max_int)
  max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

XSetTile(display,gc,tiles[0]);
draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
  tiles[4],tiles[0],max_str,"0","bswin/s vs. bswot/s",-5,0);
draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
XSetTile(display,gc,tiles[1]);
for(i=1;i<sa.index;i++)
  { /* draw bswin chart */
  barlength = length*((syss+i)->bswin)/max_int;
  XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
  }

pos_x += 2*length;
draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
XSetTile(display,gc,tiles[3]);
for(i=1;i<sa.index;i++)
  { /* draw bswot chart */
  barlength = length*((syss+i)->bswot)/max_int;
  XFillRectangle (display,pixmap,gc,pos_x-barlength,pos_y+(i-1)*y_offset,barlength,10);
  }

/* draw pswch/s  */
pos_y += i*y_offset+80;
pos_x -= 2*length;
max_int = 100;
strcpy(max_str,"100");
strcpy(mid_str,"50");
scaler_int = max_int;
max_int1 = (syss+1)->pswch;
for(i=2;i<sa.index;i++)
  if((syss+i)->pswch > max_int1)
    max_int1 = (syss+i)->pswch;
if(max_int1 > max_int)
  max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

XSetTile(display,gc,tiles[0]);
draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
  tiles[4],tiles[4],mid_str,max_str,"pswch/s",-5,35);
draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
XSetTile(display,gc,tiles[4]);
for(i=1;i<sa.index;i++)
  { /* draw pswch chart */
  barlength = 2*length*((syss+i)->pswch)/max_int;
  XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
  }
XCopyArea(display,pixmap,win,draw_para[SYSS].gc, 0,0,draw_para[SYSS].width,
  draw_para[SYSS].height, 0,0);
```

```
  draw_para[SYSS].pixmap = pixmap;
  XUndefineCursor(display,XtWindow(graph_box_globe));
} /* end of draw_syss_cb() */


/*
draw_filea_cb(), as a callback procedure, draws bar charts reflecting file access
activity. It uses the same approach as the one in draw_cpu_cb(), and thus similar in-line
comments will be omitted.
*/

void draw_filea_cb(w,canvas,cbs)
Widget w;
Widget canvas;
XmAnyCallbackStruct *cbs;
{
  Display  *display = XtDisplay(canvas);
  Window   win = XtWindow(canvas);
  XGCValues values;
  GC    gc;
  Pixmap tiles[9],pixmap;
  Arg    args[10];
  int n,i,j,x;
  int pos_x,pos_y,x_offset,y_offset,barlength;
  File_Access *filea;
  Y_axis *y_axis;
  char legend[80],mid_str[10],max_str[10];
  long length;
  int max_int,max_int1,max_int2,scaler_int;

  if(draw_para[FILEA].pixmap)
      return;
  pixmap = create_pixmap(canvas,FILEA);
  create_draw_env(canvas,tiles,&gc);
  filea = sa.filea;
  y_axis = sa.y_axis;
  length = get_display_length(display);

  /* set starting position */
  pos_x = X_START; pos_y = 70;
  XSetTile(display,gc,tiles[0]);

  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  x_offset = 2*length/10;
  y_offset = 15;
  /* determin the maximum value for drawing */
  max_int = 500;
  strcpy(max_str,"500");
  strcpy(mid_str,"");
  scaler_int = max_int;
  max_int1 = (filea+1)->iget;
  for(i=2;i<sa.index;i++)
    if((filea+i)->iget > max_int1)
      max_int1 = (filea+i)->iget;
  max_int2 = (filea+1)->namei;
  for(i=2;i<sa.index;i++)
    if((filea+i)->namei > max_int2)
      max_int2 = (filea+i)->namei;
  if(max_int2 > max_int1)
    max_int1 = max_int2;
  if(max_int1 > max_int)
    max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
    tiles[4],tiles[0],max_str,"0","iget/s and namei/s",-5,0);
  draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[0]);
  for(i=1;i<sa.index;i++)
    { /* draw iget chart */
    barlength = length*((filea+i)->iget)/max_int;
    XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
    }

  pos_x += 2*length;
  draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[1]);
  for(i=1;i<sa.index;i++)
```

```
    { /* draw namei chart */
    barlength = length*((filea+i)->namei)/max_int;
    XFillRectangle (display,pixmap,gc,pos_x-barlength,pos_y+(i-1)*y_offset,barlength,10);
    }

  /* x and y positions of next chart */
  pos_y += i*y_offset+80;
  pos_x -= 2*length;
  /* draw dirbk/s */
  max_int = 500;
  strcpy(max_str,"500");
  strcpy(mid_str,"250");
  scaler_int = max_int;
  max_int1 = (filea+1)->iget;
  for(i=2;i<sa.index;i++)
    if((filea+i)->iget > max_int1)
        max_int1 = (filea+i)->iget;
  if(max_int1 > max_int)
      max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

  XSetTile(display,gc,tiles[0]);
  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
      tiles[4],tiles[4],mid_str,max_str,"dirbk/s",-5,55);
  draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[2]);
  for(i=1;i<sa.index;i++)
      { /* draw dirbk chart */
      barlength = 2*length*((filea+i)->dirbk)/max_int;
      XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
      }

  XCopyArea(display,pixmap,win,draw_para[FILEA].gc, 0,0,draw_para[FILEA].width,
    draw_para[FILEA].height, 0,0);
  draw_para[FILEA].pixmap = pixmap;
  XUndefineCursor(display,XtWindow(graph_box_globe));
} /* end of draw_filea_cb() */


/*
draw_q_cb(), as a callback procedure, draws bar charts reflecting queue_length activity.
It uses the same approach as the one in draw_cpu_cb(), and thus similar in-line comments
will be omitted.
*/

void draw_q_cb(w,canvas,cbs)
Widget w;
Widget canvas;
XmAnyCallbackStruct *cbs;
{
  Display  *display = XtDisplay(canvas);
  Window   win = XtWindow(canvas);
  XGCValues values;
  GC    gc;
  Pixmap tiles[9],pixmap;
  Arg    args[10];
  int n,i,j,x,tile_index=0;
  int pos_x,pos_y,x_offset,y_offset,barlength;
  Queue_Length *q;
  Y_axis *y_axis;
  char legend[80],mid_str[10],max_str[10];
  long length;
  int max_int,max_int1,max_int2,scaler_int;

  if(draw_para[Q].pixmap)
      return;
  pixmap = create_pixmap(canvas,Q);
  create_draw_env(canvas,tiles,&gc);
  q = sa.q;
  y_axis = sa.y_axis;
  length = get_display_length(display);

  /* set up starting position */
  pos_x = X_START; pos_y = 70;
  XSetTile(display,gc,tiles[0]);
  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  x_offset = 2*length/10;
```

```
  y_offset = 15;
  /* determin the maximum value for drawing */
  max_int = 100;
  strcpy(max_str,"100");
  strcpy(mid_str,"");  scaler_int = max_int;
  max_int1 = (q+1)->runq_sz;
  for(i=2;i<sa.index;i++)
    if((q+i)->runq_sz > max_int1)
      max_int1 = (q+i)->runq_sz;
  max_int2 = (q+1)->swpq_sz;
  for(i=2;i<sa.index;i++)
    if((q+i)->swpq_sz > max_int2)
      max_int2 = (q+i)->swpq_sz;
  if(max_int2 > max_int1)
    max_int1 = max_int2;
  if(max_int1 > max_int)
    max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
    tiles[4],tiles[0],max_str,"0","runq-sz vs. swpq-sz",-5,0) ;
  draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[0]);
  for(i=1;i<sa.index;i++)
    { /* draw runq_sz chart */
    barlength = length*((q+i)->runq_sz)/max_int;
    XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
    }
  pos_x += 2*length;
  draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[2]);
    for(i=1;i<sa.index;i++)
    { /* draw swpq_sz chart */
    barlength = length*((q+i)->swpq_sz)/max_int;
    if(barlength <0)
      continue; /* skip zero bar length */
    XFillRectangle (display,pixmap,gc,pos_x-barlength,pos_y+(i-1)*y_offset,barlength,10);
    }

  /* x and y positions of next chart */
  pos_y += i*y_offset+80;
  pos_x -= 2*length;
  /* draw %runocc  and %swpocc */
  XSetTile(display,gc,tiles[0]);
  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
    tiles[4],tiles[0],"100","0","%runocc vs. %swpocc",-5,0);
  draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[1]);
  for(i=1;i<sa.index;i++)
    { /* draw runocc chart */
    barlength = length*((q+i)->runocc)/100;
    XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
    }

  pos_x += 2*length;
  draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[3]);
  for(i=1;i<sa.index;i++)
    { /* draw swpocc chart */
    barlength = length*((q+i)->swpocc)/100;
    if(barlength <0)
      continue;
    XFillRectangle (display,pixmap,gc,pos_x-barlength,pos_y+(i-1)*y_offset,barlength,10);
    }

  XCopyArea(display,pixmap,win,draw_para[Q].gc,
    0,0, draw_para[Q].width, draw_para[Q].height, 0,0);
  draw_para[Q].pixmap = pixmap;
  XUndefineCursor(display,XtWindow(graph_box_globe));
} /* end of draw_q_cb() */


/*
draw_tbs_cb(), as a callback procedure, draws bar charts reflecting tabel status. It uses
the same approach as the one in draw_cpu_cb(), and thus similar in-line comments will be
omitted.
*/
```

```
void draw_tbs_cb(w,canvas,cbs)
Widget w;
Widget canvas;
XmAnyCallbackStruct *cbs;
{
  Display  *display = XtDisplay(canvas);
  Window   win = XtWindow(canvas);
  XGCValues values;
  GC       gc;
  Pixmap tiles[9],pixmap;
  Arg    args[10];
  int n,i,j,x,tile_index=0;
  int pos_x,pos_y,x_offset,y_offset,barlength;
  Table_Status *tbs;
  Y_axis *y_axis;
  char legend[80],size_str[80],mid_str[10],max_str[10];
  long length;
  extern struct _choosen choosen;
  int max_int,max_int1,max_int2,scaler_int;

  if(draw_para[TBS].pixmap)
    return;
  pixmap = create_pixmap(canvas,TBS);
  create_draw_env(canvas,tiles,&gc);
  tbs = sa.tbs;
  y_axis = sa.y_axis;
  length = get_display_length(display);

  /* set up starting position */
  pos_x = X_START; pos_y = 70;
  XSetTile(display,gc,tiles[0]);
  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  x_offset = 2*length/10;
  y_offset = 15;
  to_ascii((tbs+1)->proc_tsz,size_str,0);
  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
    tiles[4],tiles[0],size_str,"0","proc-sz and proc-ov",-8,0);

  if(choosen.num >1) /* more than one item choosen */
    draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index-1);
  else /* only one item choosen */
  draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset, sa.index);
  XSetTile(display,gc,tiles[tile_index]);
  tile_index++;
  for(i=1;i<sa.index;i++)
      { /* draw proc_sz chart */
      barlength = length*((tbs+i)->proc_sz)/(tbs+1)->proc_tsz;
      XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
      }
  pos_x += 2*length;
  if( choosen.num >1)
      draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index-1);
  else
      draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[tile_index]); tile_index++;
  for(i=1;i<sa.index;i++)
      { /* draw proc_ov chart */
      barlength = length*((tbs+i)->proc_ov)/(tbs+1)->proc_tsz;
      XFillRectangle (display,pixmap,gc,pos_x-barlength,
        pos_y+(i-1)*y_offset,barlength,10);
      }

  /* x and y positions of next chart */
  pos_y += i*y_offset+80;
  pos_x -= 2*length;
  /* draw inod-sz and inod-ov */
  XSetTile(display,gc,tiles[0]);
  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  to_ascii((tbs+1)->inod_tsz,size_str,0);
  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
      tiles[4],tiles[0],size_str,"0","inod-sz and inod-ov",-8,0);
  if( choosen.num >1)
      draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index-1);
  else
      draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index) ;
  XSetTile(display,gc,tiles[tile_index]);
  tile_index++;
```

```
    for(i=1;i<sa.index;i++)
        { /* draw inod_sz chart */
        barlength = length*((tbs+i)->inod_sz)/(tbs+1)->inod_tsz;
        XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
        }
    pos_x += 2*length;
    if( choosen.num >1)
        draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index-1);
    else
        draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
    XSetTile(display,gc,tiles[tile_index]); tile_index++;
    for(i=1;i<sa.index;i++)
        { /* draw inod_ov chart */
        barlength = length*((tbs+i)->inod_ov)/(tbs+1)->inod_tsz;
        XFillRectangle (display,pixmap,gc,pos_x-barlength,
          pos_y+(i-1)*y_offset,barlength,10);
        }

    /* draw file-sz  and file-ov */
    pos_y += i*y_offset+80;
    pos_x -= 2*length;
    XSetTile(display,gc,tiles[0]);
    draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
    to_ascii((tbs+1)->file_tsz,size_str,0);
    draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
        tiles[4],tiles[0],size_str,"0","file-sz and file-ov",-8,0);
    if( choosen.num >1)
        draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index-1);
    else
        draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
    XSetTile(display,gc,tiles[tile_index]); tile_index++;
    for(i=1;i<sa.index;i++)
        { /* draw file_sz chart */
        barlength = length*((tbs+i)->file_sz)/(tbs+1)->file_tsz;
        XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
        }
    pos_x += 2*length;  if( choosen.num >1)
        draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index-1);
    else
        draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
    XSetTile(display,gc,tiles[tile_index]); tile_index++;
    for(i=1;i<sa.index;i++)
        { /* draw file_ov chart */
        barlength = length*((tbs+i)->file_ov)/(tbs+1)->file_tsz;
        XFillRectangle (display,pixmap,gc,pos_x-barlength,
          pos_y+(i-1)*y_offset,barlength,10);
        }

    /* draw lock-sz */
    pos_y += i*y_offset+80;
    pos_x -= 2*length;
    XSetTile(display,gc,tiles[0]);
    draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
    to_ascii((tbs+1)->lock_tsz,size_str,0);
    to_ascii((tbs+1)->lock_tsz/2,mid_str,0);
    draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
        tiles[4],tiles[4],mid_str,size_str,"lock-sz",-5,50);
    if( choosen.num >1)
        draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index-1);
    else
        draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
    XSetTile(display,gc,tiles[tile_index]); tile_index++;
    for(i=1;i<sa.index;i++)
        { /* draw lock_sz chart */
        barlength = length*((tbs+i)->lock_sz)/(tbs+1)->lock_tsz;
        XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
        }

    XCopyArea(display,pixmap,win,draw_para[TBS].gc,
        0,0, draw_para[TBS].width,draw_para[TBS].height, 0,0);
    draw_para[TBS].pixmap = pixmap;
    XUndefineCursor(display,XtWindow(graph_box_globe));
} /* end of draw_tbs_cb() */


/*
draw_msg_cb(), as a callback procedure, draws bar charts reflecting semaphore and message
```

```
activity. It uses the same approach as the one in draw_cpu_cb(), and thus similar in- line
comments will be omitted.
*/

void draw_msg_cb(w,canvas,cbs)
Widget w;
Widget canvas;XmAnyCallbackStruct *cbs;
{
  Display  *display = XtDisplay(canvas);
  Window   win = XtWindow(canvas);
  XGCValues values;
  GC    gc;
  Pixmap tiles[9],pixmap;
   Arg    args[10];
  int n,i,j,x,tile_index=0;
  int pos_x,pos_y,x_offset,y_offset,barlength;
  Msg_Sema *msg;
  Y_axis *y_axis;
  char legend[80],mid_str[10],max_str[10];
  long length;
  int max_int,max_int1,max_int2,scaler_int;

  if(draw_para[MSG].pixmap)
    return;
  pixmap = create_pixmap(canvas,MSG);
  create_draw_env(canvas,tiles,&gc);
  msg = sa.msg;
  y_axis = sa.y_axis;
  length = get_display_length(display);
  /* set up starting position */
  pos_x = X_START; pos_y = 70;
  XSetTile(display,gc,tiles[0]);
  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  x_offset = 2*length/10;
  y_offset = 15;
  /* determin the maximum value for drawing */
  max_int = 500;
  strcpy(max_str,"500");
  strcpy(mid_str,"");
  scaler_int = max_int;
  max_int1 = (msg+1)->msge;
  for(i=2;i<sa.index;i++)
    if((msg+i)->msge > max_int1)
      max_int1 = (msg+i)->msge;
  max_int2 = (msg+1)->sema;
  for(i=2;i<sa.index;i++)
    if((msg+i)->sema > max_int2)
      max_int2 = (msg+i)->sema;
  if(max_int2 > max_int1)
    max_int1 = max_int2;
  if(max_int1 > max_int)
    max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
     tiles[4],tiles[0],max_str,"0","msg/s vs. sema/s",-5,0);
  draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index) ;
  XSetTile(display,gc,tiles[tile_index]);
  tile_index++;
  for(i=1;i<sa.index;i++)
    { /* draw message chart */
    if((msg+i)->msge == 0.0)
      continue;
    barlength = length*((msg+i)->msge)/max_int;
    if(barlength <1)
      continue; /* skip if length < 1 */
    XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
    }
  pos_x += 2*length;
  draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[tile_index]); tile_index++;
  for(i=1;i<sa.index;i++)
    { /* draw sema chart */
    if((msg+i)->sema == 0.0)
      continue;  /* skip if sema == 0 */
    barlength = length*((msg+i)->sema)/max_int;
    if(barlength <1)
      continue;
```

```
    XFillRectangle (display,pixmap,gc,pos_x-barlength,
    pos_y+(i-1)*y_offset,barlength,10);
    }
  XCopyArea(display,pixmap,win,draw_para[MSG].gc,
    0,0, draw_para[MSG].width, draw_para[MSG].height, 0,0);
  draw_para[MSG].pixmap = pixmap;
  XUndefineCursor(display,XtWindow(graph_box_globe));
} /* end of draw_msg_cb() */


/*
draw_p_cb(), as a callback procedure, draws bar charts reflecting paging activity. It uses
the same approach as the one in draw_cpu_cb(), and thus similar in-line comments will be
omitted.
*/

void draw_p_cb(w,canvas,cbs)
Widget w;
Widget canvas;
XmAnyCallbackStruct *cbs;
{
  Display  *display = XtDisplay(canvas);
  Window   win = XtWindow(canvas);
  XGCValues values;
  GC      gc;
  Pixmap tiles[9],pixmap;
  Arg    args[10];
  int n,i,j,x,tile_index=0;
  int pos_x,pos_y,x_offset,y_offset,barlength;
  Paging *p;
  Y_axis *y_axis;
  char legend[80],mid_str[10],max_str[10];
  long length;
  int max_int,max_int1,max_int2,scaler_int;

  if(draw_para[P].pixmap)
    return;
  pixmap = create_pixmap(canvas,P);
  create_draw_env(canvas,tiles,&gc);
  p = sa.p;
  y_axis = sa.y_axis;
  length = get_display_length(display);

  /* set up starting position */
  pos_x = X_START; pos_y = 70;
  XSetTile(display,gc,tiles[0]);
  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  x_offset = 2*length/10;
  y_offset = 15;
  /* determin the maximum value for drawing */
  max_int = 500;
  strcpy(max_str,"500");
  strcpy(mid_str,"");
  scaler_int = max_int;
  max_int1 = (p+1)->vflt;
  for(i=2;i<sa.index;i++)
    if((p+i)->vflt > max_int1)
      max_int1 = (p+i)->vflt;
  max_int2 = (p+1)->pflt;
  for(i=2;i<sa.index;i++)
    if((p+i)->pflt > max_int2)
      max_int2 = (p+i)->pflt;
  if(max_int2 > max_int1)
    max_int1 = max_int2;
  if(max_int1 > max_int)
    max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
    tiles[4],tiles[0],max_str,"0","vflt/s vs. pflt/s",-5,0);
  draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[tile_index]); tile_index++;
  for(i=1;i<sa.index;i++)
      {  /* draw vflt chart */
      barlength = length*((p+i)->vflt)/max_int; XFillRectangle (display,pixmap,gc,pos_x,
      pos_y+(i-1)*y_offset,barlength,10);
      }
  pos_x += 2*length;
```

```
  draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[tile_index]); tile_index++;
  for(i=1;i<sa.index;i++)
      { /* draw pflt chart */
      barlength = length*((p+i)->pflt)/max_int;
      XFillRectangle (display,pixmap,gc,pos_x-barlength,
        pos_y+(i-1)*y_offset,barlength,10);
      }
  /* x and y positions of the next chart */
  pos_y += i*y_offset+80;
  pos_x -= 2*length;
  /* draw pgfil and rclm */
  max_int = 500;
  strcpy(max_str,"500");
  strcpy(mid_str,"");
  scaler_int = max_int;
  max_int1 = (p+1)->pgfil;
  for(i=2;i<sa.index;i++)
      if((p+i)->pgfil > max_int1)
         max_int1 = (p+i)->pgfil;
  max_int2 = (p+1)->rclm;
  for(i=2;i<sa.index;i++)
      if((p+i)->rclm > max_int2)
         max_int2 = (p+i)->rclm;
  if(max_int2 > max_int1)
      max_int1 = max_int2;
  if(max_int1 > max_int)
      max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);
  XSetTile(display,gc,tiles[0]);
  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
      tiles[4],tiles[0],max_str,"0","pgfil/s vs. rclm/s",-5,0);
  draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[tile_index]); tile_index++;
  for(i=1;i<sa.index;i++)
      { /* draw pgfil chart */
      barlength = length*((p+i)->pgfil)/max_int;
      XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
      }
  pos_x += 2*length;
  draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[tile_index]); tile_index++;
  for(i=1;i<sa.index;i++)
      { /* draw rclm chart */
      barlength = length*((p+i)->rclm)/max_int;
      XFillRectangle (display,pixmap,gc,pos_x-barlength,
        pos_y+(i-1)*y_offset,barlength,10);
      }
  XCopyArea(display,pixmap,win,draw_para[P].gc,
      0,0, draw_para[P].width, draw_para[P].height, 0,0);
  draw_para[P].pixmap= pixmap;
  XUndefineCursor(display,XtWindow(graph_box_globe));
} /* end of draw_p_cb() */


/*draw_mem_blk_cb(), as a callback procedure, draws bar charts reflecting memory block
activity. It uses the same approach as the one in draw_cpu_cb(), and thus similar in-line
comments will be omitted.
*/

void draw_mem_blk_cb(w,canvas,cbs)
Widget w;
Widget canvas;
XmAnyCallbackStruct *cbs;
{
  Display  *display = XtDisplay(canvas);
  Window   win = XtWindow(canvas);
  XGCValues values;
  GC    gc;
  Pixmap tiles[9],pixmap;
  Arg    args[10];
  int n,i,j,x,tile_index=0;
  int pos_x,pos_y,x_offset,y_offset,barlength;
  Mem_Blk *mem_blk;
  Y_axis *y_axis;
  char legend[80],mid_str[10],max_str[10];
```

```
  long length;
  int max_int,max_int1,max_int2,scaler_int;

  if(draw_para[MEM_BLK].pixmap)
      return;
  pixmap = create_pixmap(canvas,MEM_BLK);
  create_draw_env(canvas,tiles,&gc);
  mem_blk = sa.mem_blk;
  y_axis = sa.y_axis;
  length = get_display_length(display);

  /* set up starting position */
  pos_x = X_START; pos_y = 70;
  XSetTile(display,gc,tiles[0]);
  draw_rectangle(display,pixmap,gc,pos_x,pos_y,2*length,sa.index*15);
  x_offset = 2*length/10;
  y_offset = 15;
  /* determin the maximum value for drawing */
  max_int = 1000000;
  strcpy(max_str,"1000000");
  strcpy(mid_str,"");
  scaler_int = max_int /2;
  max_int1 = (mem_blk+1)->freemem;
  for(i=2;i<sa.index;i++)
    if((mem_blk+i)->freemem > max_int1)
      max_int1 = (mem_blk+i)->freemem;
  max_int2 = (mem_blk+1)->freeswp;
  for(i=2;i<sa.index;i++)
    if((mem_blk+i)->freeswp > max_int2)
      max_int2 = (mem_blk+i)->freeswp;
  if(max_int2 > max_int1)
    max_int1 = max_int2;
  if(max_int1 > max_int)
    max_int = reset_max_int(max_int1,scaler_int,mid_str,max_str);

  draw_scaler(display,pixmap,gc,pos_x,x_offset,pos_y,
    tiles[4],tiles[0],max_str,"0","freemem vs. freeswp",-20,0);
  draw_time(display,pixmap,gc,pos_x-90,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[tile_index]); tile_index++;
  for(i=1;i<sa.index;i++)
    { /* draw freemem chart */
    barlength = length*((mem_blk+i)->freemem)/max_int;
    XFillRectangle (display,pixmap,gc,pos_x,pos_y+(i-1)*y_offset,barlength,10);
    }

  pos_x += 2*length;
  draw_time(display,pixmap,gc,pos_x+15,pos_y,y_offset,sa.index);
  XSetTile(display,gc,tiles[tile_index]); tile_index++;
  for(i=1;i<sa.index;i++)
    { /* draw freeswp chart */
    barlength = length*((mem_blk+i)->freeswp)/max_int;
    XFillRectangle (display,pixmap,gc,pos_x-barlength, pos_y+(i-1)*y_offset,barlength,10);
    }

  XCopyArea(display,pixmap,win,draw_para[MEM_BLK].gc,
    0,0, draw_para[MEM_BLK].width, draw_para[MEM_BLK].height, 0,0);
  draw_para[MEM_BLK].pixmap = pixmap;
  XUndefineCursor(display,XtWindow(graph_box_globe));
} /* end of draw_mem_blk_cb() */


/*
get_display_length() gets display height and converts it into
mm unit. It returns the converted value.
*/

long get_display_length(dpy)
Display *dpy;
{
   int myscreen=DefaultScreen(dpy); /* get screen pointer */
   long pixels,mm,length;

   pixels =  DisplayHeight (dpy, myscreen);
   mm = DisplayHeightMM (dpy,myscreen);
   length = ( (50*pixels) + (mm/2) ) / mm;
   return length;
} /* end of get_display_length() */
```

```
/*
draw_rectangle() draws a rectangle for bar chart.
*/

void draw_rectangle(dpy,pixmap,gc,x,y,width,height)
Display *dpy;
Pixmap pixmap;
GC gc;
int x,y,width,height;
{
  XDrawRectangle(dpy,pixmap,gc,x,y,width,height);
}


/*
draw_scaler() draws sclaer for the bar chart */
*/

void draw_scaler(dpy,pixmap,gc,x,x_offset,y,dot_tile,
          black_tile,mid,end,title,mid_offset,title_offset)
Display *dpy;
Pixmap pixmap;
GC gc;
int x,x_offset,y;
Pixmap dot_tile,black_tile;
char *title,*mid,*end;
int mid_offset,title_offset;
{
  int i;

  XSetTile(dpy,gc,dot_tile);
  for(i=1;i<10;i++)
    {
    if(i==5) /* middle point */
         {
         XSetTile(dpy,gc,black_tile); /* set divider */
         XDrawLine(dpy,pixmap,gc,x+x_offset*i,y,
           x+x_offset*i,y+sa.index*15);
        XSetTile(dpy,gc,dot_tile);
         }
    else /* draw scaler line */
         XDrawLine(dpy,pixmap,gc,x+x_offset*i,y,x+x_offset*i,y+sa.index*15);
    if(i==5)
         {
         XDrawImageString(dpy,pixmap,gc,x+x_offset*i-8+mid_offset,y-15,mid,strlen(mid));
         XDrawImageString(dpy,pixmap,gc,x+x_offset*i-80+title_offset,y-45,title,
           strlen(title));
         }
    } /* for */
  /* draw title for x axis */
  XDrawImageString(dpy,pixmap,gc,x+x_offset*i-5,y-15,end,strlen(end));
  XDrawImageString(dpy,pixmap, gc, x-2,y-15,"0",strlen("0") );
} /* end of draw_scaler() */


/*
draw_time() draws time for y axis in the bar chart.
*/

void draw_time(dpy,pixmap,gc,x,y,y_offset,index)
Display *dpy;
Pixmap  pixmap;
GC gc;
int x,y,y_offset,index;
{
   int i;
   Y_axis *y_axis;
   y_axis = sa.y_axis;

   if(!y_axis)
        { printf("\nNull y_axis!\n"); exit(-1); }
   XDrawImageString(dpy,pixmap,gc,x+15,y-15,"time",strlen("time"));
   for(i=1;i<index;i++)
        XDrawImageString(dpy,pixmap,gc,x,y+(i-1)*y_offset+10,
          y_axis[i].time,strlen(y_axis[i].time));
} /* end of draw_time() */
```

```
/*
make_legend() concatenates strings for a legend title.
*/

void make_legend(legend,name,value)
char *name;
char *legend;
char *value;
{
   strcpy(legend,name);
   strcat(legend," ");
   strcat(legend,value);
} /* end of make_legend() */


/*
quit_callback(), as a callback procedure, pops down the help box if it is popped up and
then pops down the widget passed to it.
*/

void quit_callback(w,form,call_data)
Widget w;
Widget form;
XmAnyCallbackStruct *call_data;
{
  if(XtIsManaged(help_box))
    XtUnmanageChild(help_box);
  XtUnmanageChild(form);
} /* end of quit_callback() */


/*
unset_state_2(), as a callback procedure, sets toggle button state to insensitive.
*/

static void unset_state_2(w,bt,call_data)
Widget w;
Widget bt;
XmAnyCallbackStruct *call_data;
{
    XmToggleButtonSetState(bt,False,False);
} /* end of unset_state_2() */


/*
to_ascii() converts an int value  to its ASCII
representation.
*/

void to_ascii(value,string,index)
int value;
char *string;
int index;
{
  int sign = value; /* get sign */

  string[index++] = (absa(value) % 10) + '0';
  value /= 10; /* removes the least significant digit */
  if(absa(value) >0)
    to_ascii(value,string,index);
  else
    {
    if( sign <0)
         string[index++] = '-'; /* place the minus */
    string[index] = NULL;     /* append NULL to the string */
    str_rev(string,0,index-1); /* reverse the string */
    }
} /* end of to_ascii() */


/*
str_rev() reverses the contents of the character string.
*/

void str_rev(string,start,end)
char* string;
```

```
int start, end;
{
  int length; /* length of the string */
  char temp;  /* tempory storage for the character swap */

  length = strlen(string);
  if(start >= length)
    return;
   /* assign end to the last character in the string */
  else if( end >= length)
    end = length -1;
  if(start >= end)
    return;             /* the string is reversed */
  else                  /* otherwise swap the letters and * /
    {                   /* invoke str_rev() recursively */
    temp = string[start];
    string[start] = string[end];
    string[end] = temp;
    str_rev(string, ++start, --end);
    }
} /* end of str_rev() */


/*
reset_max_int() determines and returns the maximum value,
and converts the value into strings for chart scaler title.
*/

int reset_max_int(real,scaler_int,mid_str,max_str)
int real,scaler_int;
char *mid_str,*max_str;
{
    int max;

    max = ( real + 2*scaler_int - real%scaler_int );
    to_ascii(max/2,mid_str,0); /* middle string */
    to_ascii(max,max_str,0);
    return max;
} /* end of reset_max_int() */


/*
unmap_dev_cmd_cb() pops down the widget window associated with the device activity menu.
*/

void unmap_dev_cmd_cb(w,whichone,cbs)
Widget w;
int whichone;
XmAnyCallbackStruct *cbs;
{
  XtUnmanageChild(Device_Menu_Command);
} /* end of unmap_cmd_cb() */



/* help.text starts here */
#0
     Report CPU utilization:

     %usr, %sys, %wio, %idle - portion of time running in
     user mode, running in system mode, idle with some
     process waiting for block I/O, and otherwise idle.
#0
#1
     Report buffer activity:

     bread/s, bwrit/s - transfers per second of data between
     system buffers and disk or other block devices;

     lread/s, lwrit/s - the number per second of logical
     read and write requests issued by the system to block
     devices;

     %rcache, %wcache - cache hit ratios, i. e.,
     (1-bread/lread) as a percentage;

     pread/s, pwrit/s - the number per second of read and write
     requests issued by the system to raw (physical) devices.
```

```
#1
#2
     Report activity for each block device such as a disk or
     tape drive.  When data is displayed, the device
     specification dsk- is generally used to represent a
     disk drive.  The device specification used to represent
     a tape drive is machine dependent.  The following
     activity data is reported:

     %busy, avque - portion of time device was busy
     servicing a transfer request, average number of
     requests outstanding during that time;

     r+w/s, blks/s - number of data transfers from or to
     device, number of bytes transferred in 512-byte units;

     avwait, avserv - average time in ms. that transfer
     requests wait idly on queue, and average time to be
     serviced (which for disks includes seek, rotational
     latency and data transfer times).
#2
#3
     Report TTY device activity:

     rawch/s, canch/s, outch/s - input character rate, input
     character rate processed by canon, output character
     rate;

     rcvin/s, xmtin/s, mdmin/s - receive, transmit and modem
     interrupt rates.
#3
#4
     Report system calls:

     scall/s - system calls of all types;

     sread/s, swrit/s, fork/s, exec/s - specific system
     calls;

     rchar/s, wchar/s - characters transferred by read and
     write system calls.
#4
#5
     Report system swapping and switching activity:

     swpin/s, swpot/s, bswin/s, bswot/s - number of
     transfers and number of 512-byte units transferred for
     swapins and swapouts (including initial loading of some
     programs);

     pswch/s - process switches, which happen when (1) A system
     call lead to a road block, (2) An interrupt was invoked resulting
     in awakening a higher priority process, or (3) A one second clock
     interrupt is invoked.
#5
#6
     Report use of file access system routines:

     iget/s - the number of an OS routine iget() invoked per second for
     locating the inode entry of a file (i-number);

     namei/s - the number of an OS routine namei() invoked per second for
     searching the various directory files to get the associated
     i-number of a file corresponding to a special path;

     dirblk/s - the number of an OS routine dirblk() invoked per second
     for counting directory block reads issued by the system.
#6
#7
     Report average queue length while occupied, and % of
     time occupied:

     runq-sz, %runocc - run queue of processes in memory and
     runnable;

     swpq-sz, %swpocc - swap queue of processes swapped out
     but ready to run.
#7
#8
     Report status of process, inode, file tables:

     text-sz, proc-sz, inod-sz, file-sz, lock-sz -
```

```
     entries/size for each table, evaluated once at sampling
     point;

     ov - overflows that occur between sampling points for
     each table.
#8
#9
     Report message and semaphore activities:

     msg/s - message sending and receiving times per second
     sema/s - semaphore operations per second.
#9
#10
     Report paging activities:

     vflt/s - address translation page faults (valid page
     not in memory);

     pflt/s - page faults from protection errors (illegal
     access to page) or "copy-on-writes" (always 0 under
     DYNIX);

     pgfil/s - vflt/s satisfied by page-in from filesystem;

     rclm/s - valid pages reclaimed for free list (always 0
     under DYNIX).
#10
#11
     Report unused memory pages and disk blocks:

     freemem - average pages available to user processes;

     freeswap - disk blocks available for process swapping.
#11
```

VITA

Haibo Du

Candidate for the Degree of

Master of Science

Thesis: DISPLAY OF SEQUENT SYMMETRY S/81 PERFORMANCE USING X WINDOW SYSTEM FACILITIES

Major Field: Computer Science

Biographical:

Personal Data: Born in Shanghai, People's Republic of China, March 2, 1958, the son of Qingru Du and Xiaohu Liu. Married to Hanzhen Yang on November, 1986.

Education: Graduated from High School No. 62, Shanghai, People's Republic of China, in February 1976; received Bachelor of Arts in English Language and Literature from East China Normal University, People's Republic of China, in July 1986; received the Master of Science degree in Occupational and Adult Education from Oklahoma State University in May 1990; completed requirements for the Master of Science degree in Computer Science at Oklahoma State University in May 1993.

Professional Experience: Research Assistant, School of Occupational and Adult Education, Oklahoma State University, August 1988 to May 1992.