REALTIME GRAPHICAL DISPLAY OF SYSTEM MEASUREMENTS

By

SYED NASIR RAZA

Master of Science

in Applied Physics

University of Karachi

Karachi, Pakistan

1991

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
December 1997

REALTIME GRAPHICAL DISPLAY OF

SYSTEM MEASUREMENTS

Thesis Approved

Mansur Samadzadeh

Thesis Advisor

Blayne E. Mayfield

J. Chandler

Wayne B Powell

Dean of the Graduate College

PREFACE

An operating system performs its job poorly when a subset of its resources, such as CPU, memory, or I/O devices, is heavily loaded. When the number and/or capacity of the resources cannot keep up with the demand, some user applications are forced to wait, resulting in a longer average turnaround time and a generally slower system. It is the job of a system manager or administrator to keep the overall system running efficiently by tuning and optimizing the performance of the system. System administrators use system performance commands or software tools to find out how a system is behaving at any given time. For example, performance monitoring and display tools can be used to determine why the CPU utilization is low or whether thrashing exists in the system and as a consequence the system is spending more time doing paging than performing computations. This thesis reports on the design and implementation of a real-time system performance software tool on a Sequent Symmetry S/81. This tool uses a Graphical User Interface (GUI) that can display various system parameters to help determine what to do to improve system performance. The tool provides a dynamic graphical interface for existing textual performance capture/reporting tools.

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to and thank my thesis advisor Dr. Mansur H. Samadzadeh, who motivated me by his valuable instruction and example throughout my thesis research work. Dr. Samadzadeh gave generously of his valuable time and expertise for correcting my work and rendering helpful suggestions.

I also extend my sincere appreciation to Drs. Blayne E. Mayfield and John P. Chandler for their advice and willingness to serve on my graduate committee. Their suggestions and support were very helpful throughout the study.

Also, I would like to express my gratitude to my parents, brothers, sister, and all my friends who were mentally with me all the time.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF FIGURES

FIGURE PAGE

CHAPTER I

INTRODUCTION

In order to make informed system tuning decisions, one must know what the system is doing in term of the attributes of the running processes and the characteristics of the resources being used.

System performance monitoring tools assist system administrators in keeping a system running efficiently by tuning it. Performance tools also help users make the best use of a system by user-level scheduling [Thobani and Samadzadeh 92]. They can help system administrators in deciding what action should be taken if something goes wrong.

It is clear that if one particular job is running slowly, it does not necessarily mean that the system's performance is bad. A particular job can generally be made to run faster by changing its algorithm, data structures, or both. However, to improve the overall system throughput, we have to structure the system so that its resources are used efficiently and fairly among all users [SunSoft 93]. Improving system performance involves customizing the kernel (see Section 2.2 for explanation) which is forbidden territory [Loukides 90]. However, simple techniques (such as running memory intensive jobs at non-peak hours) can be used to improve a system's behavior.

There are many different factors that play a vital role in determining a system's behavior. Such factors include the number and the nature of the processing units, usage

patterns, I/O configuration, and software consideration (e.g., compilers and editors). A slow system may be made to yield a better performance by changing one or more of these factors. It is also known that optimizing one aspect of a system's performance may lead to a negative impact on one or more of the other aspects of the system [Loukides 90]. For example, to minimize the internal fragmentation of memory in a paged memory environment, we may decrease the page size, which results in larger page table sizes and consequently larger number of I/O operations.

By using a system performance tool, once it is detected that a part of a system is overloaded, some measure can be taken to overcome the problem.

The Sequent Symmetry S/81 uses the DYNIX/ptx operating system which is compatible with the AT&T UNIX System V3.2 [Sequent 91]. On Sequent, there are a number of commands that can help determine a picture of the system's activities. Such commands include **monitor** (monitor system's activity), **sar** (system activity report), and **ps** (process status). The **monitor** command is the only utility that gives a comprehensive picture of the overall system activity, including all its active processors, by using a set of bar graphs. The **monitor** command reports on various system parameters that include total user time, total system time, number of processes, and number of context switches.

The main objective of this thesis was to implement an interactive graphical user interface by using X Window System facilities to display graphically what the **monitor** command provides textually.

Chapter II of this thesis report provides a literature review and context for Sequent S/81, UNIX kernel, Curses, System performance commands, Graphical User Interface, X Window, and OSF/Motif programming. Chapter III describes the implementation part of

this thesis. Finally, Chapter IV contains the summary of the thesis and some suggestions for future work.

CHAPTER II

LITERATURE REVIEW

This chapter contains a brief introduction to the Sequent S/81 machine, UNIX kernel, Curses, system performance commands, graphical user interface, X Window system, and OSF/Motif programming.

2.1 Sequent Symmetry S/81 Architecture

Sequent Symmetry S/81 is one of Sequent's mainframe class multiprocessor computer systems. The Sequent S/81 runs both DYNIX V3.0 and DYNIX/ptx operating systems. Sequent's implementations of the UNIX operating system is binary compatible with the operating systems on the other Symmetry family of computer systems [Symmetry 90].

2.2 UNIX Kernel

The UNIX operating system consists of two main parts: system routines and kernel. The kernel consists of everything between the system call interface and the physical hardware [Stevens 90]. It provides the file system, CPU scheduling, memory

management, and I/O management [Silberschatz 94]. When a system initially boots, it initiates the first process and is a base that enables everything else to work [Miscovich and Simon 94]. Because the kernel is the main controlling program, it is always resident in the physical memory and it cannot be swapped out to disk like other programs.

2.3 Curses

Curses is the UNIX library of functions for controlling the terminal screen from a C program [Rochkind 88]. By using curses, we can output characters to any point on the screen, do it in the highlight mode, etc. Curses is a package of different routines on UNIX that control cursor movements and windowing [Schreiner 90]. These routines help to improve the screen driver interface for a program or to improve a program's user interface. Optimized cursor motion, physical terminal independence, multiple windows, and video attributes are some of the benefits that curses offers a C programmer [Strang 91]. Because of these advantages, curses can be used to write programs that use the entire screen to present information in an attractive and organized way, instead of sequential and line-oriented output at the bottom of the screen. The e-mail utility called **pine** [Sequent 91] uses curses to display its output on the screen.

2.4 System Performance and Relevant Commands

A system's performance is generally poor when the system feels sluggish to the users. When the system seems slow or when jobs seem to take longer than they should to

complete, the system administrator should start considering ways to improve the system's performance.

System performance commands or software tools provide a means of checking a system's status at any time. These commands or software tools retrieve system information either already stored in a log file or from the live (i.e., running) system. These tools collect the necessary data from various data structures in the kernel memory. Besides reading from the kernel memory, these tools also collect useful information about I/O devices, CPU utilization, and memory utilization.

The first statistic that a system administrator should look at, when the performance is poor, is the system load average. Checking the system load average provides a general view of the system's activity. In the UNIX environment, the average number of processes in the kernel's run queue during an arbitrary interval is called load average [Loukides 90]. The run queue contains processes that are not waiting for any external event (such as a mouse click), are not waiting by running the wait loop, and are not stopped by having pressed the CTRL and Z buttons simultaneously [Loukides 90].

Although a load average command (such as the **xload** command) is convenient and informative, it may not give an accurate picture of a system's load. This is because the jobs that are waiting for disk I/O are also counted as runnable jobs by the load average command [Loukides 90]. And, if the Network File System does not respond for some reason, a process may have to wait for hours for the I/O to complete. This lack of response results in an increase in the load average, even when the system is not really doing any more work [Frisch 91]. Another problem with the load average command is

that it does not differentiate between low and high priority jobs, because low priority jobs may be swapped out to disk when jobs with higher priority enter the system.

On BSD and System V UNIX, the **uptime** command can be used to check the system load average over the last minute, the last 5 minutes, or the last 15 minutes [Loukides 90]. This command is useful for checking to see whether the load average is climbing or falling. On Sequent S/81, the **xload** command can be used to check the system's load visually using X Window System utilities.

There are many different general purpose commands or software tools that can be used to determine a system's status.

2.4.1 **cron** Command

The clock daemon (**cron**) executes UNIX commands at specified dates and times. The data for the commands are stored in files that are referred to as crontab files. The **cron** command reads the crontab files and runs the commands. The **cron** command can be used to gather data regularly and to clean up the file system. This command is available on all UNIX systems. Its synopsis is:

/etc/**cron** [-m maxjobs] [-q]

where

-m maxjobs — This option modifies the maximum number of **cron** jobs that are running on the system concurrently. The default value of maxjobs is 200. This flag can be used to help control **cron** job traffic and system load.

-q — This option suppresses job rescheduling messages because if a new **cron** job starts when the maximum number of jobs are already running, the new job will be rescheduled and a message will be printed.

2.4.2 **iostat** Command

The I/O status command provides information about disk usage. Besides disk data, **iostat** gives some important information about CPU usage also. This command is for BSD UNIX only. Its synopsis is:

iostat

2.4.3 **perfmeter** Command

The performance meter command provides a visual report that gives several system statistics such as CPU utilization, number of processes, and number of users. This command is available on SunOS only. Its synopsis is:

perfmeter

2.4.4 **prdaily** Command

The previous day's accounting data command provides a set of daily accounting reports. This command is available on System V UNIX only. The **prdaily** command is used to format a report of the previous day's accounting data. This report resides in /usr/adm/acct/sum/rprt mmdd where mmdd is the month and day of the report. Its synopsis is:

/usr/lib/acct/**prdaily** [-l] [-c] [mmdd]

The current daily accounting reports may be printed by typing **prdaily**. Previous days' accounting reports can be printed by using the mmdd option and specifying the exact report date desired. The -l flag gives a report of usage by login ID for the specified date. The -c flag gives a report of resource usage by command.

2.4.5 **ps** Command

The process status command reports information about the processes running, sleeping, or waiting in the system. This command is available on all UNIX systems. Its synopsis is:

ps [options]

The **ps** command prints certain information about active processes. If we ignore the options, information is printed about processes associated with the controlling terminal. The **ps** command accepts a large number of options including the following:

-e	Prints information about every process currently running.
-d	Prints information about all processes except process group leaders.
-a	Prints information about all processes most frequently requested.
-f	Prints a full listing about processes.
-F	Prints a full listing about all processes.
-l	Prints a long listing.
-t termlist	This option prints information about the processes that are associated with the terminals given in termlist.
-p proclist	This option prints information about the processes whose process ID numbers are given in proclist.
-u uidlist	This option prints information about the processes whose user ID numbers or login names are given in uidlist.
-g grplist	This option prints information about the processes whose process group ID numbers appear in grplist.

2.4.6 **sa** Command

The system accounting command provides a set of accounting reports showing the commands that have executed and the resources that they required. This command is available on BSD UNIX only. Its synopsis is:

sa [dd]

where dd represents the day of the current month.

2.4.7 **sar** Command

The system activity report command provides information about CPU usage, I/O information, user time, system time, idle time, page faults, etc. This command is available on System V UNIX only. Its synopsis is:

sar [-abcdmpqruvwyA] interval number

sar [-abcdmpqruvwyA] -f file

The first command samples system activity at every interval (specified in seconds) where the number of samples is indicated by the number field. The second command gathers data from a file indicated by the file field. These data files reside in /usr/adm/sa/sadd where dd indicates the day of the month. The options are defined as follows [Loukides 90]:

a Reports usage of file access system calls.

b Reports buffer cache usage and hit rate.

c Reports about the system calls.

d Reports block device activity.

m Reports message and semaphore activity.

p Reports paging activity.

q Reports average run queue length and average swap queue length.

r Reports about the unused memory pages and disk blocks.

u Reports CPU utilization.

v Reports status of system tables.

w Reports swapping and paging activity.

y Reports terminal activity.

A Reports all data, same as all the above options [-abcdmpqruvwyA].

2.4.8 **uptime** Command

The system's up time command shows how long system has been up as well as the system's load average. This command is available on BSD and System V UNIX. Its synopsis is:

uptime

2.4.9 **xload** command

The system load command displays a periodically updating histogram of system load average using the X Window system. Its synopsis is:

xload [-hl color] [-label string] [-scale integer] [-update seconds]

where

-hl color This option specifies the color of the scale lines.

-label string	This option takes a string to put as a label of xload window.
-scale integer	This option specifies the minimum number of tick marks in the histogram.
-update second	This option specifies the update rate of the xload display in seconds.

Among these software tools or commands, **ps** and **sar** are comparatively more powerful. They can give comprehensive reports about the performance of the system.

2.5 Graphical User Interface

A large amount of research has been performed to determine what type of user interface would be most useful and suitable to various target audiences from computer programmers to computer users. The results of these research efforts indicate that a Graphical User Interface or GUI (pronounce "gooey") should be used to allow users to manipulate objects on the screen in much the same way that users manipulate objects on their desks [Flanagan 94]. When a user interface makes use of graphical objects like windows and menus, it is said to be a graphical user interface [Barkakati 91]. If GUI is used for designing an interface, the resulting system would be generally more intuitive and easier to learn and use. The GUI of an application describes how an application appears on the screen (the look) and how the user interacts with it (the feel) [Heller 91]. The user interacts with the application by typing at the keyboard, by clicking the mouse's buttons, and selecting and dragging various graphic elements of the application with the mouse.

2.6 X Window System

The X Window system is a computer based windowing system which allows the users to write programs that use a Graphical User Interface or GUI.

2.6.1 Definition

The X Window system came into being as a result of the Athena project at MIT [Sebern 94]. At that time, MIT had a large number of different computers, all running under different operating systems. The purpose of the Athena project was to link them together in such a way that they all would become accessible over the network and each computer would become a network resource.

The basic goal of the Athena project was to add graphics capabilities such as windows, menus, and buttons to the X system as a network resource, so that an application can prepare graphics at any location on the network and both send and receive input from a user elsewhere on the network. According to Flanagan [Flanagan 94], this technique has many advantages such as:

- A user can sit in front of a SUN workstation and interact with the graphical application by using keyboard and mouse but the actual application may be running on a remote network site.
- A user on a PC with a graphics card and mouse can interacts with a database program on a mainframe.
- A demonstration program can be executing on one machine and be displayed on a number of graphical display terminals on remote sites.

The X Window works on a client-server model [Du 93]. A client is any application running anywhere on the network, and the server is a graphical display

terminal. The client and server run as separate processes in a multiprocess environment and they communicate using a high-level X protocol [Ali and Yang 94]. If the client and the server are running on the same UNIX machine, they may use UNIX streams or sockets. If the application (client) is running on one UNIX machine and the server is running on another, they may communicate over Ethernet or Optical cable using the TCP/IP protocols.

The client and server communicate by sending messages to one another. There are three kind of messages [Nye and O'Reilly 90]:

A one-way request: This request goes from the application (client) to the server. Such a request is to open a window, change a background color, or move a window. Such request allow the application to continue processing without delay.

A round-trip request: This request may be sent from the client to the server and this type of request always returns from the server to the client. Such requests are to find the position of the mouse cursor or the number of colors of the window. These kind of requests may be slow in a heavily loaded network.

Events: There are different kinds of events that are sent from the server to the client. These events are generated and convey different information about the application. For instance, events are generated when the mouse crosses the boundary of a window, when a key is pressed or released, or when part of a window is exposed. The user can also generate these events when they move the mouse over a window and handle the keyboard. Each window of an application can receive only interested events and reject unwanted events.

The greatest advantage of the X Window system is that the application is not machine dependent [Smith 94]. If the client and the server are communicating using the X protocol, the client can be any machine on the network.

2.6.2 Multiple Software Layers

The basic tasks of X Window can be define as follows [Heller 91]:

1. Establish a network connection between an application and the X server.
2. Mapping and displaying windows on the server.
3. Drawing graphics and displaying text.
4. Receiving and responding to events.
5. Dealing with multiple window system.

These tasks are carried out by different software layers (as shown in Figure 1) that are used by the X Window system.

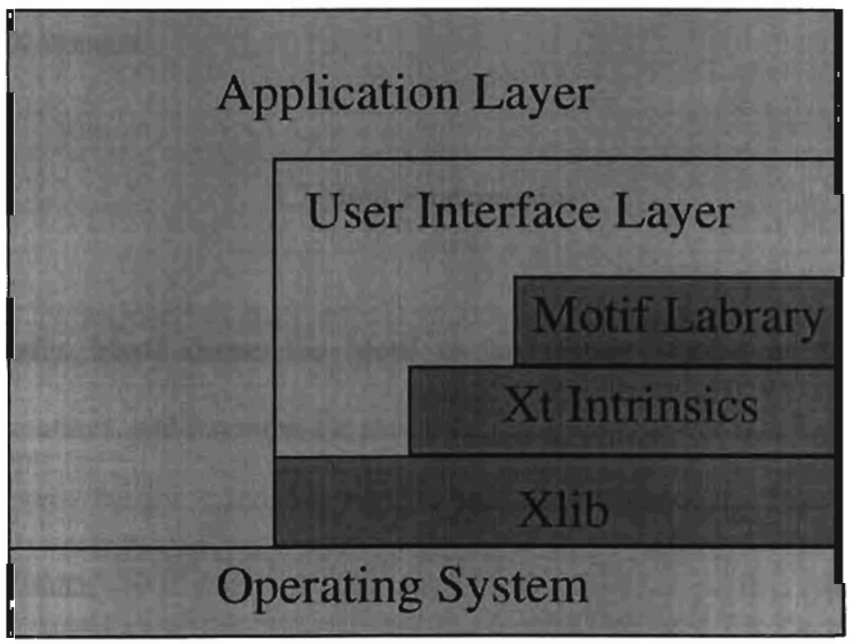


Figure 1. Layers of X Windows (Source: [Heller 91])

The bottom layer consist of Xlib which is a set of library routines that implement the basic X calls. Xlib routines work at the lowest level and they include building blocks for

any type of windowed application. The Xlib contains nearly 300 functions that have been preprogrammed to create, move, resize, stack, and destroy windows, and to draw rectangles, arc, lines, and graphics images [Nye and O'Reilly 90]. The next layer (above the Xlib layer) is called toolkit or Xt that gives the structure and organization of an application to the user. The toolkit is divided into two levels, the first level deals with the objects in a window system such as windows, buttons, and pull-down menus. These objects within X Window are called widgets (see Section 2.7.6 for explanation). The second level is called Intrinsics or Xt Intrinsics. This level is responsible for dealing with object and user interactions. The highest layer is the application layer. This layer may be a mixture of Xlib, Intrinsics, and widgets. The application layer may interacts with all layers of the X Window.

2.7 Motif Programming

Generally, Motif means the Motif toolkit which contains a set of widgets, convenience routines, and functions for manipulating graphical user interface objects.

2.7.1 Brief History

Motif was developed by OSF (Open Software Foundation) [George and Riches 94]. In 1988, the Apollo Computer Inc., DEC, Groupe Bull, Hewlett-Packard, IBM Corp., Nixdorf Computer AG, N.V., and Siemens AG worked together to sponsor a nonprofit corporation called Open Software Foundation. According to Sebern [Sebern 94], the basic goals of OSF are: (1) Portability - which allow applications to run on multiple

different systems, and (2) Interoperability - which allow different systems to work together transparently.

2.7.2 Definition

Motif is a set of widgets that is available for the X Window system developed by OSF [OSF 91a]. Motif provides a graphical user interface that offers PC style behavior [OSF 91b]. According to Heller [Heller 91], OSF/Motif offers:

- A common user interface across various platforms.
- An application programming interface which is widely available.
- A three-dimensional (3D) look for window objects.
- An easy-to-use presentation manager.

2.7.3 Motif Style Guide

The Motif style guide provides the look and feel of an application and it is used by widget programmers, window manager developers, and application programmers. If a programmer want to create a new widget, (s)he should follow the style guide. The style guide has a number of standards including the following [OSF 91b]:

mwm: The Motif Window Manager allows the X Window system user to manipulate the application's main window by a set of user interface objects such as resize handles, title bar, maximize and minimize buttons and window manager button.

Toolkit: The OSF/Motif toolkit provides a standard for graphical user interfaces that can be applied to any application. This standard provides uniform look and feels that allow users to learn new applications quickly.

UIL: The User Interface language is a Motif specific programming language and provides the programmer with a tool by which the initial state of the graphical user interface can be created.

2.7.4 Motif Components

Motif uses a layered architecture [Sebern 94] as shown in Figure 2. The top layer is the application layer.

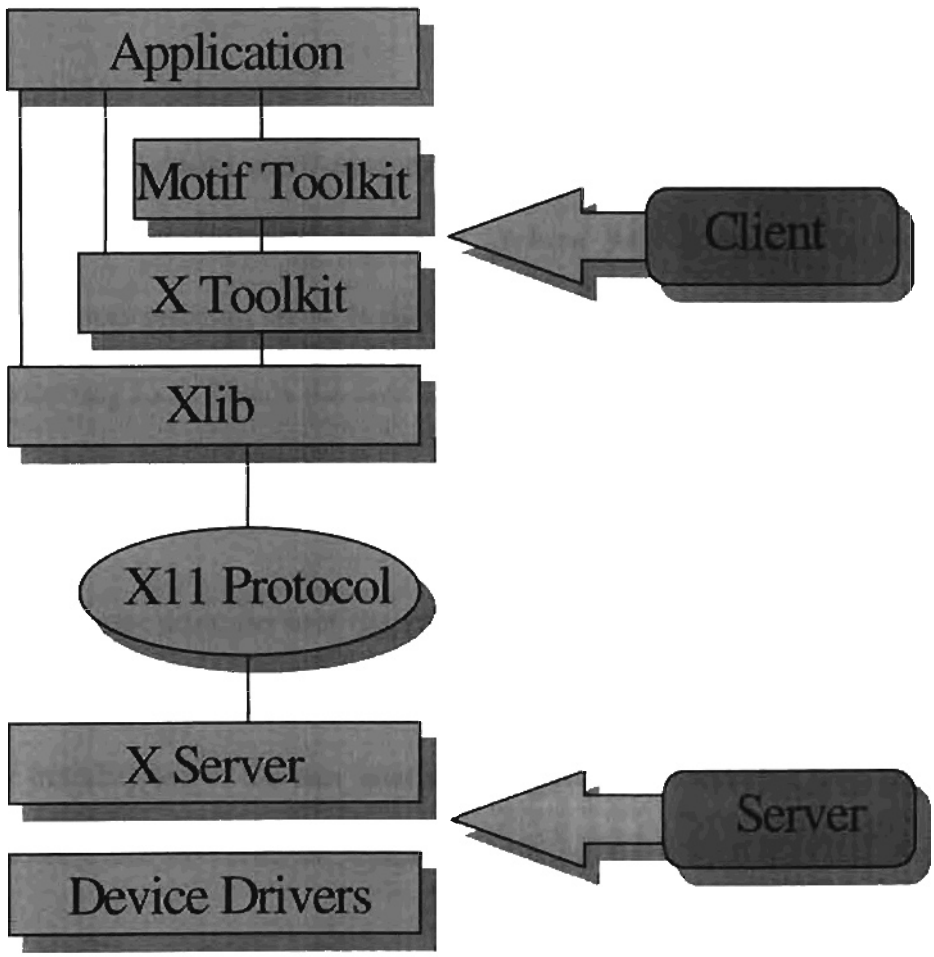


Figure 2. Motif Layered Architecture (Source: [Sebern 94])

The client component consist of Motif toolkit, X toolkit, and Xlib. The lowest layer in the client component is the Xlib that generates and sends X11 protocol requests to the server

through the communication path. The X toolkit provides a higher level programming interface. This layer is known as the Intrinsics layer. The Motif toolkit provides basic building blocks for GUI application such as widgets [Flanagan 94]. Each widget has a number of resources such as font, color, and size. The sever component is responsible for interaction with the user using keyboard, screen, and mouse. Both client and server communicate with each other using TCP/IP, DECnet, or a message passing technique.

2.7.5 Structure of Motif Applications

If we compare a Motif application with a traditional application, we find that a traditional application is sequential in nature [Sebern 94], as shown in Figure 3. Whenever a traditional program starts, it take input from the user or read a file and then goes to the processing loop. Hence the user can interact with the application only on the program's own terms. On the other hand, a Motif application is an event driven program as shown in Figure 4.

In a Motif application, the user has primary control over the application interface and the program must honor any request from the user at any time [Smith 94]. In a Motif program, after initialization of the user interface, the program enters in a loop and waits for an event to occur and then does appropriate processing corresponding to that event.

2.7.6 Widget

A widget is an abstract data type [Nye and O'Reilly 90]. It is actually a data structure defining how it will look on the screen as well as a set of procedures that will determine how it operates. Each widget belongs to a specific class of widgets that is

organized into a hierarchy. When we use any particular widget in our program, that widget inherits attributes from its parent, grandparent, and others all the way up to the root of the hierarchy. Examples of widget are Push Button, Text Label, and Frame.

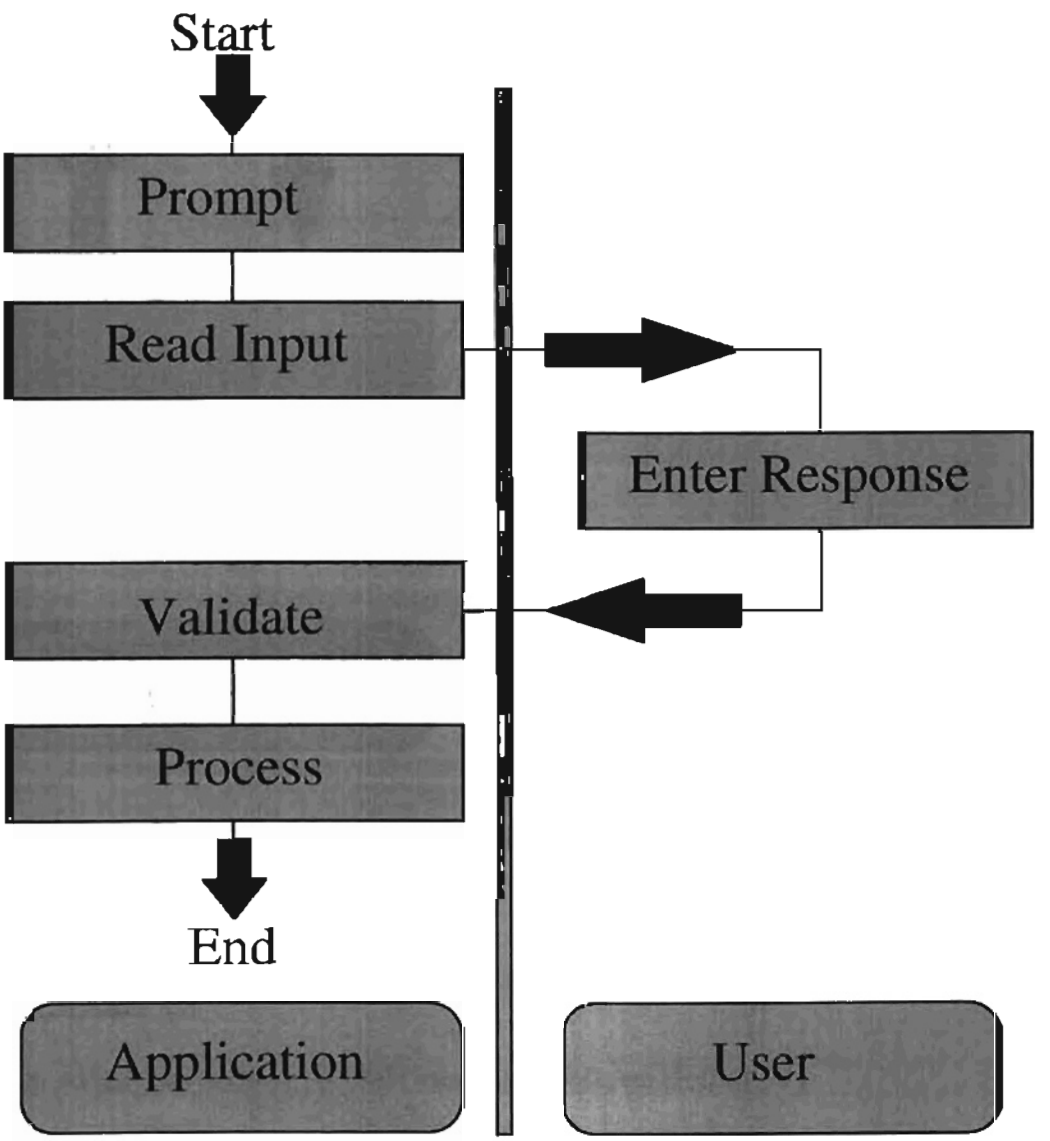


Figure 3. Traditional Application (Source: [Sebern 94])

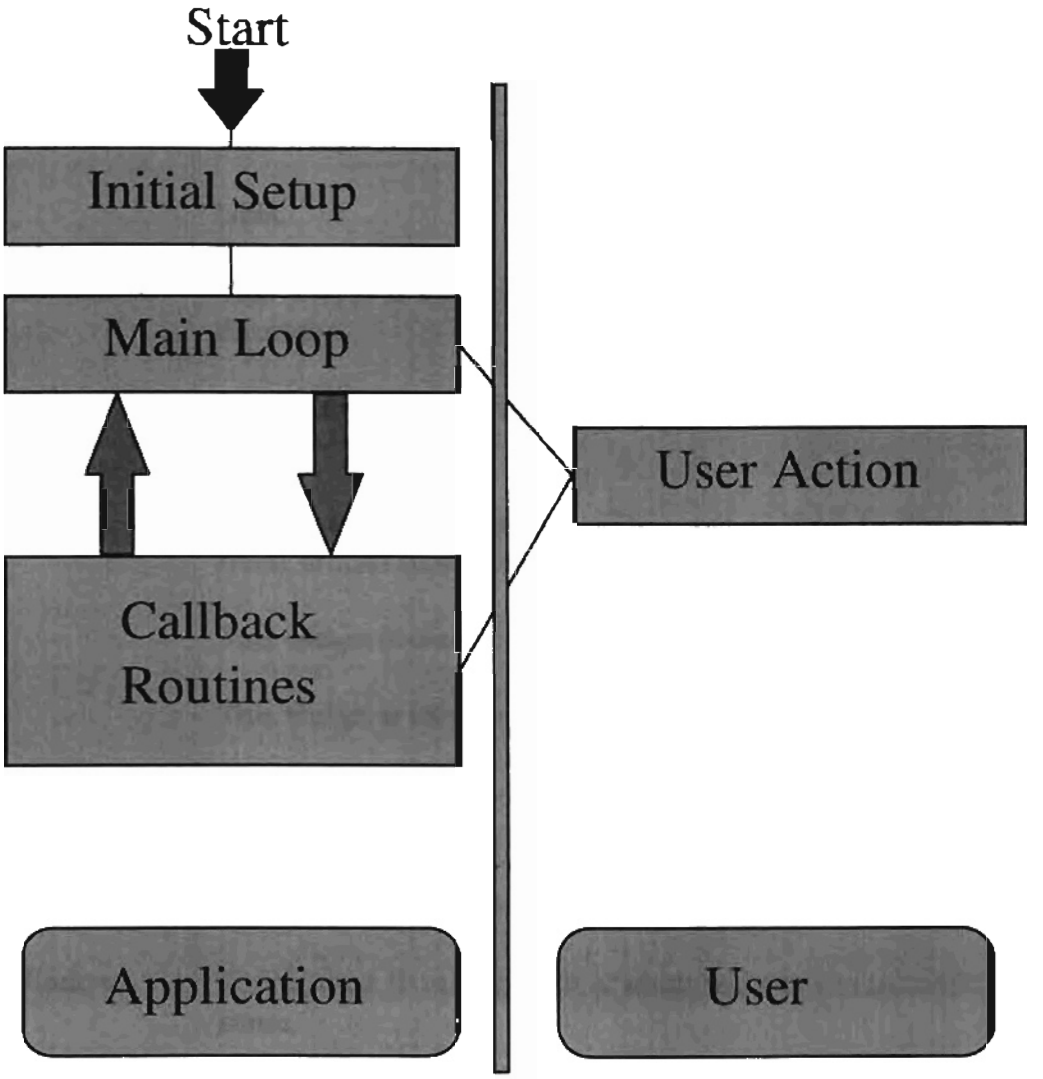


Figure 4. Motif Application (Source: [Sebern 94])

2.7.7 Motif Widget Set

The widget set defined by OSF/Motif is as follows [Heller 91]:

ArrowButton: This widget is used as a push button and it has the shape of an arrow head.

BulletinBoard: This widget is used to manage one or more widgets according to their positions.

CascadeButton: This widget is used as a push button in a pulldown menu.

DrawingArea:	This widget is used to draw graphic objects such as line, arc, and rectangle.
DrawnButton:	This widget is used as a push button and it can use a graphic as a label.
FileSelectionBox:	This widget is used to display and select a file from an indicated directory.
Form:	This widget is a container widget and is used to manage one or more widgets.
Frame:	This widget is a container widget and is used to draw a picture frame around similar widgets
Label:	This widget is used to display noneditable text as a label .
List:	This widget is used to present a list of options.
MainWindow:	This widget is a container widget and is used to manage other widgets.
MessageBox:	This is used to display pop-up messages such as an error message.
PanedWindow:	This is used to manage one or more widgets in resizeable window panes.
PushButton:	This widget is used as a push button.
RowColumn:	This is a container widget and is used to manage other widgets in equally sized and spaced rows or columns.
Scale:	This widget is used to indicate a specific numeric value.
ScrollBar:	This is used to display a page inside the window larger than the window size.
Separator:	This is used to visually separate two or more objects from other objects.
Text:	This is used to display or enter ASCII text data.
ToggleButton:	This widget is used as a push button and it has two states: normal and pressed.

2.7.8 Motif Data Type

The Motif library provides several new data types and these data types are defined in the Motif header files. Some of the OSF/Motif data types are [OSF 91a]:

XmFontList: This data type is used for font information.

XmString: This is Motif type character string and it is used to hold a character string.

XmString Direction: This data type is used to specify the printing direction of a string.

CHAPTER III

DESIGN AND IMPLEMENTATION ISSUES

This chapter describes the design and implementation issues of the **xmon** program.

3.1 Purpose

The main purpose of the real-time graphical display tool for system performance measurements, called **xmon**, is to provide a dynamic visual representation for various system performance measurements such as system load averages, number of traps, number of system calls, and CPU activities.

3.2 Program Architecture

The real-time graphical display tool or **xmon** consists four separate processes as shown in Figure 5, including the main part of the program which is responsible for its Graphical User Interface or GUI. The main program consists of Motif function calls to display various windows on the client's terminal screen to show various system performance data visually.

The other three parts are the children of the main program. These parts are considered as the backbone of the main program because they collect the various system performance data and send them to the main program via pipes as shown in Figure 5. These parts are labeled as Main Program, Child1, Child2, and Child3.

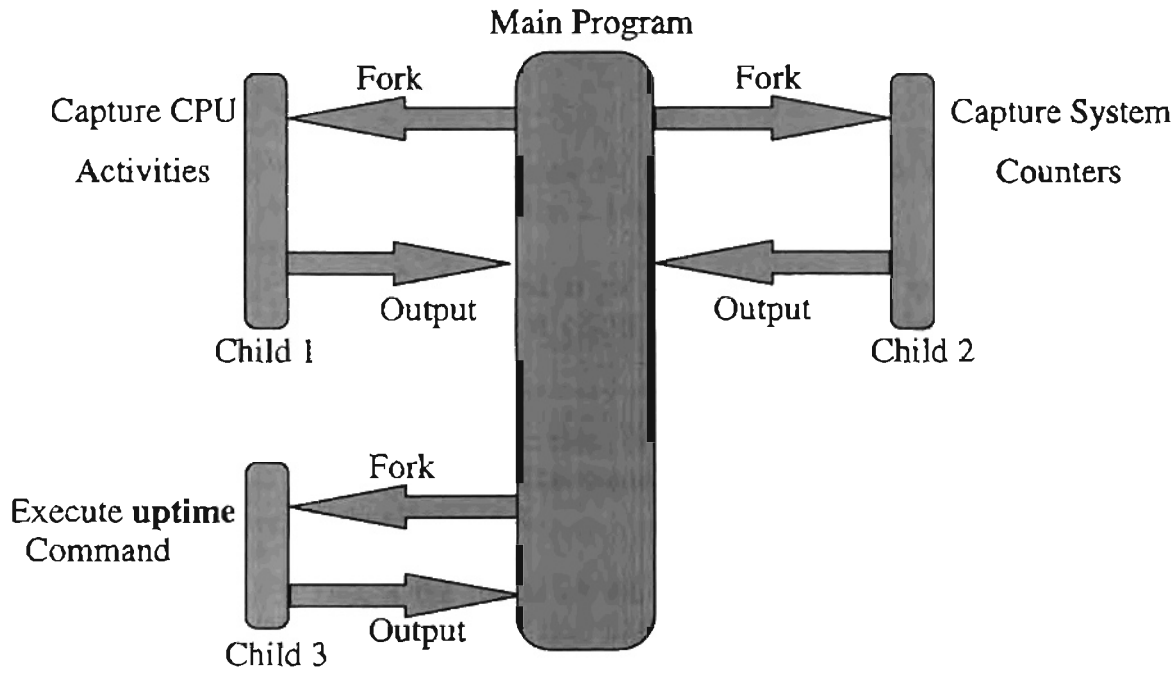


Figure 5. Block Diagram of **xmon** Program

3.3 Main Program

The main program is responsible for initiating X Window, mapping the main program window on the terminal screen, displaying system load averages, taking the user input, and generating its three children. The main program window has four buttons labeled CPU ACTIVITIES, SYSTEM COUNTERS, HELP, and EXIT. Besides these

buttons, the main window has a drawing area widget on the right side of the buttons. This area is used to draw bar graphs of the system load averages during the last one minute, the last five minutes, and the last fifteen minutes. The main window is shown in Figure 6. The four buttons are created by using Motif convenience function as described below:

Button_widget=XtVaCreateManagedWidget (button_name,

XmPushButtonWidgetClass, parent, argument_list, NULL)

where

Button_widget: Push button created by the above function whose data type is Widget (see Section 2.3 for explanation).

button_name: This name is used to get resources from the resource database for this widget and it could be any name. Its data type is character string.

XmPushButtonWidgetClass: This is the class of the widget to be created and acts as the class identifier. This name is defined in the public header file for a particular widget.

parent: This is the parent of the widget being created and it must be a manager widget that has already been created. Its data type is Widget.

argument_list: This argument list is used to set various widget resources such as widget height and width, and background and foreground color.

There are five frame widgets on the main window (as shown in Figure 7) that serve as a container of other child widgets. All four buttons are children of Frame 1. Each buttons has a callback function associated with it and each callback is registered by a toolkit function as shown below:

XtAddCallback(widget, XmNactivateCallback, function_name, NULL)

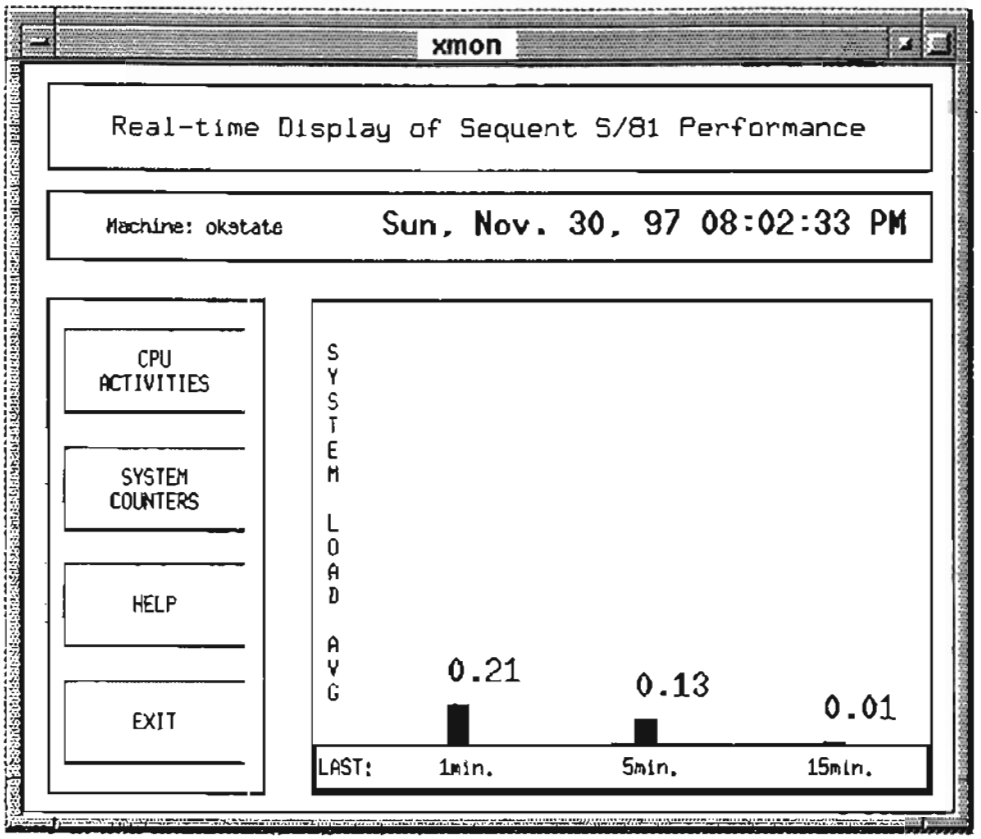


Figure 6. Main Window

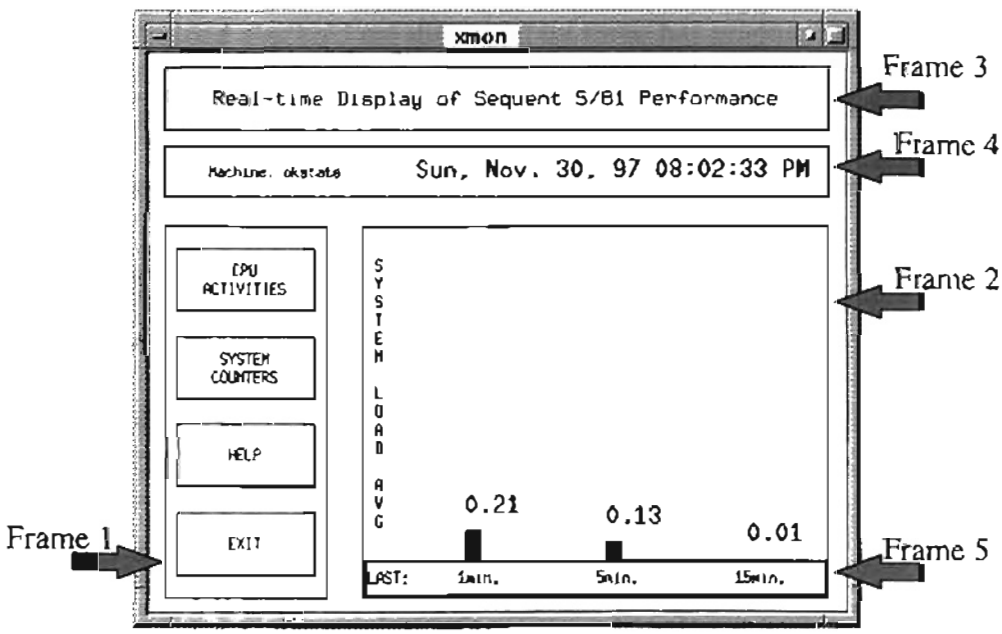


Figure 7. Frames of Main Window

where

widget: The name of the widget in which the callback is to be installed. Its data type is Widget.

XmNactivateCallback: The name of the callback resource. Each widget has a number of different callback resources. For instance, for the Arrowbutton widget, the callback resources are XmNactivatecallback, XmNarmcallback, and XmNdisarmcallback; and for the DrawingArea widget, the resources are XmNexposecallback, XmNresizecallback, and XmNinputcallback.

function_name: The function_name is the name of function that is called when callback is activated. This is the pointer to the callback function.

The callback resources for all the four buttons are the same, i.e., XmNactivateCallback. The reason being that the callback function should be activated whenever the user presses the mouse button.

3.4 Main Window Graphics

The system load averages are displayed by three bar graphs. These bar graphs are labeled "1 min.", "5 min.", and "15 min.", which means that the system load averages during the last one minute, the last five minutes, and the last fifteen minutes.

To draw bar graphs, we need the Drawing Area widget that can be obtain by calling a Motif convenience function as shown below:

```
drawing_widget=XtVaCreateManagedWidget ( widget_name,
                    XmDrawingAreaWidgetClass, parent, argument_list, NULL )
```

This drawing area is a child of Frame 2 (Figure 7).

To get the related data of system load averages, the main program forks child3 and this child executes the **uptime** command (see Section 2.4.8) to get system load

averages. The output of child3 goes to the main program through a pipe as shown in Figure 5.

Frame 3 (Figure 7) contains the title string of the main window and Frame 4 contains two different widgets:

1. The label widget that contains the name of the machine (okstate).
2. The Drawing Area widget that displays the current date and time.

The reason for having a Drawing Area widget instead of a normal label widget (to show the date and time) is that the time is changing every second and the time should be displayed on the widget every second. This means that the widget label must be updated every second. But changing the widget's label every second causes the flickering effect on the widget and consequently the entire label cannot be seen clearly. By using Drawing Area widget the flickering effect can be eliminated. This widget is useful if the graphics needs to be updated very fast.

3.5 CPU Activities Window

When the user clicks on the button labeled CPU ACTIVITIES (Figure 6), the CPU activities window pops up showing activities of the 24 processors on the Sequents S/81. To display the user time and the system time graphically, scale widget were used. The scale widget can be created by using:

```
scale_widget=XtVaCreateManagedWidget ( widget_name,
        xmScaleWidgetClass. parent, resource values pair, NULL )
```

Each processor is represented by two scale widgets, one for displaying system time and the other one for displaying user time. They are labeled by 'S' for system time and by 'U' for user time, as shown in Figure 8. There are 48 scale widgets in this window. Each scale widget shows the percentage of CPU time spent in user mode and in system mode.

The CPU activities window has a push button labeled CPU REPORT located at the top right of the screen. This button is used to generate a report about CPU utilization. When the user clicks on this button, a window pops up showing the relative CPU utilization of the 24 processors on the Sequent S/81 using bar graphs as shown in Figure 9. This window also informs the user as to which CPU is utilized most heavily and which one is utilized most lightly during the execution of the **xmon** program.

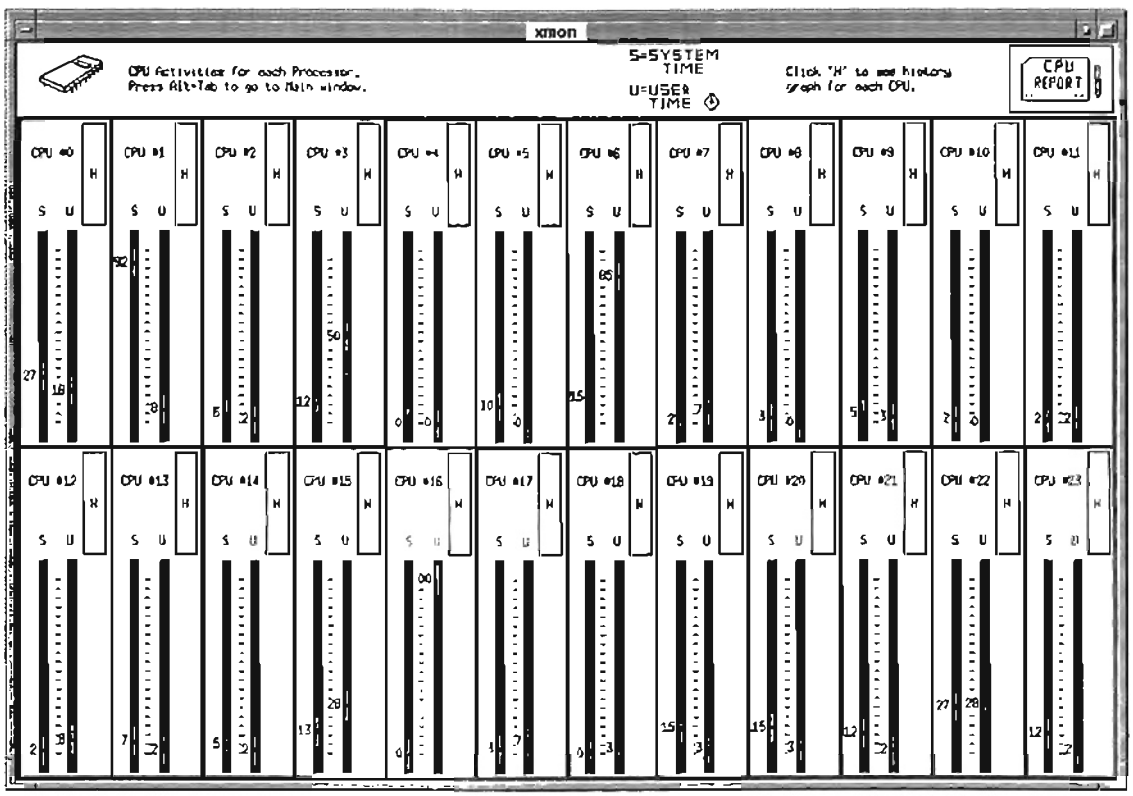


Figure 8. CPU Activities Window

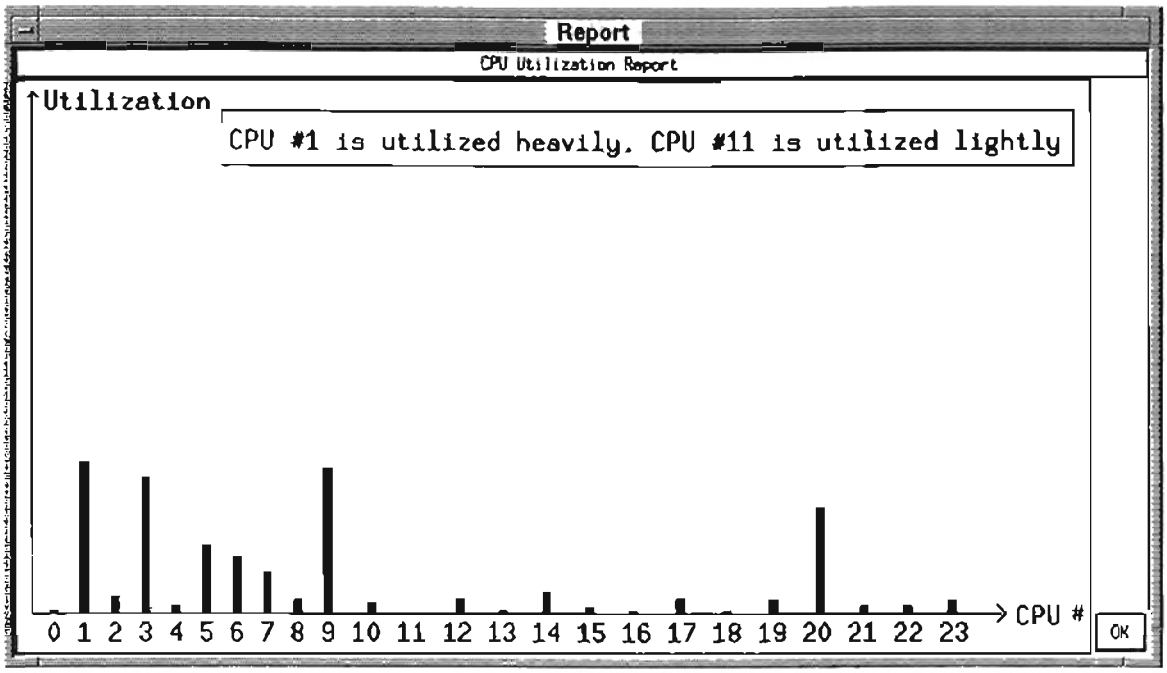


Figure 9. CPU Report Window

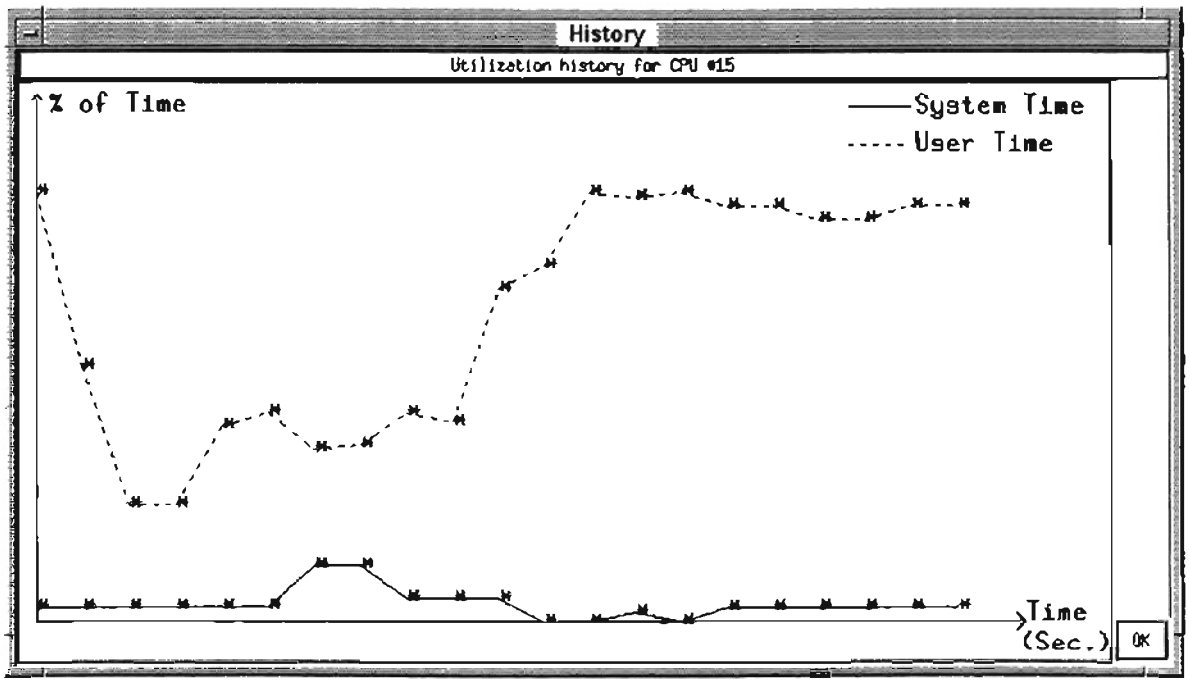


Figure 10. CPU History Window

Each microprocessor box in the CPU activities window has a history button labeled H. When the user clicks on that button, a window pops up and displays the history of the selected CPU in terms of system time and user time, as illustrated in Figure 10. This window has two different graphs in different colors. The system time is displayed by solid line colored red and the user time is displayed by dashed line colored green.

3.6 System Counters Window

When the user clicks on the button labeled SYSTEM COUNTER (Figure 6), a window pops up showing various system counters on Sequent S/81, as shown in Figure 11.

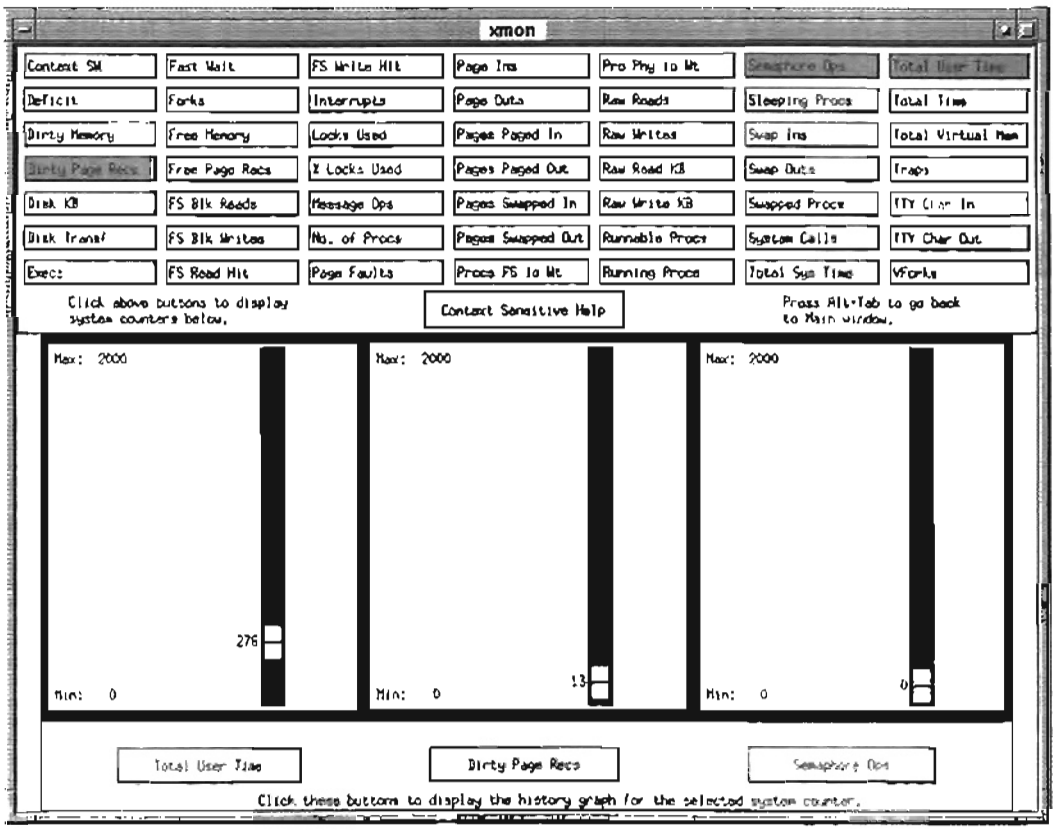


Figure 11. System Counters Window

This window has 50 push buttons including the context sensitive help button. By using these buttons, the user can observe the activities of various system counters (see Section 3.7 for explanation) such as number of traps per second, number of page faults per second, and number of context switches per second.

To present the values of various system counters in real time, scale widgets were used. Each counter is represented by one scale widget. When the user presses any of the counter buttons, the selected system counter appears on the left side of the bottom part of the window and the previously displayed three counters shift right.

By default, initially the system counter window displays three system counters: Total User Time, Dirty Page Recs, and Semaphore Ops (Figure 11). Each displayed system counter has a default minimum and maximum range which is 0 to 2000. During the execution of the **xmon** program, whenever the value of a displayed counter exceeds 2000, the maximum range becomes the new value plus 20% of the new value.

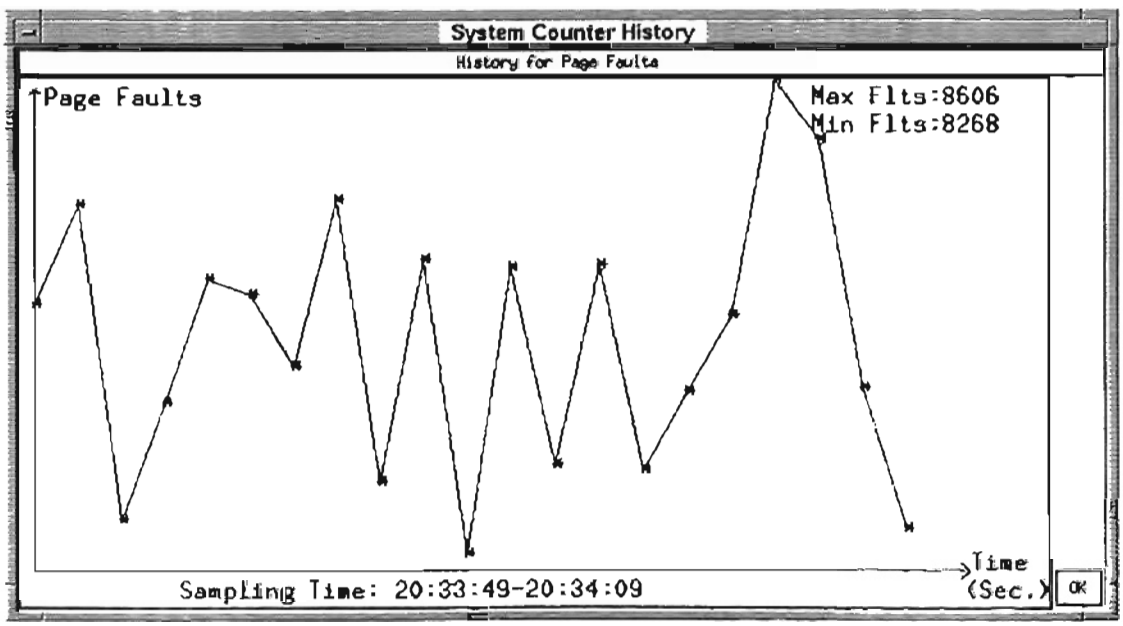


Figure 12. History Window for a System Counter or percentage of progress

The user may also observe the history of any system counter by using the three buttons located at the bottom of the window (Figure 11). When any of the buttons get pressed, a window pops up and presents the history of the selected system counter as illustrated in Figure 12. By observing the history of a system counter in term of a line graph, a user can determine easily whether the value of the counter is increasing or decreasing with the passage of time.

The system counter window also provides context sensitive help. This help is activated by pressing a button labeled "Context Sensitive Help" located in the center of the screen after the system counter buttons.

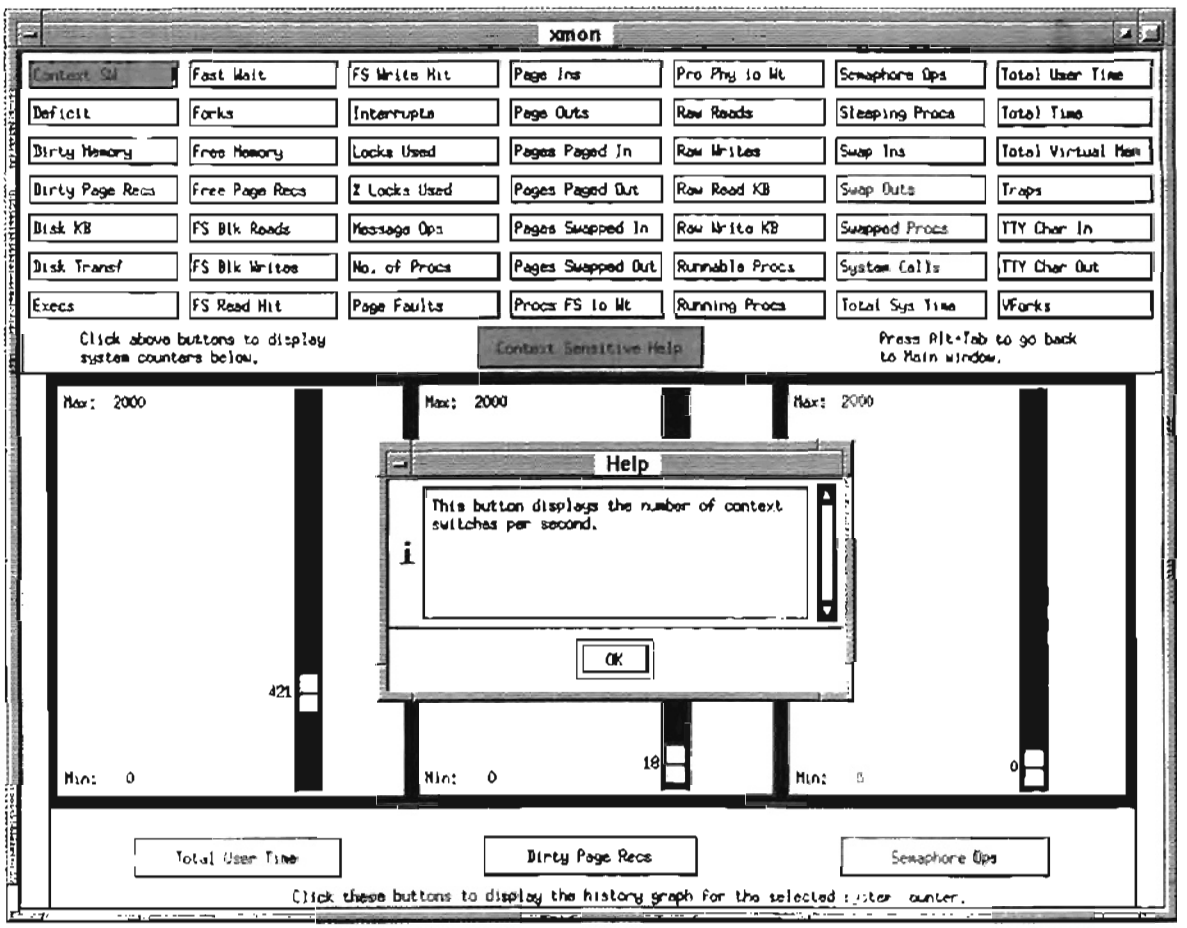


Figure 13. Context Sensitive Help Window

When the user presses the context sensitive help button, the cursor changes its shape to a "hand cursor", which indicates that the **xmon** program is now in the context sensitive help mode. When the user presses any system counter button in this mode, a window pops up and displays a help text about the selected system counter as shown in Figure 13.

3.7 Definition of System Counters

The kernel memory keeps a number of different system counters for the purpose of system administration. These counters are incremented or decremented as various system actions occur. The Sequent S/81 has the following system counters [Sequent 91]:

Context Switches:

Number of context switches per second.

Deficit:

Virtual memory deficit in Kbytes.

Dirty Memory:

Dirty memory in Kbytes.

Dirty Page Recs:

Number of page reclaims from the dirty list per second. The sum of page reclaims from the dirty list and the free list equals the total reclaims.

Disk Kbytes:

Number of Kbytes transferred per second.

Disk Transfers:

Number of disk transfers per second.

Execs:

Number of execs calls per second.

Fast Wait:

Number of processes that are waiting for disk I/O, page I/O, or a kernel resource.

Forks:

Number of forks per second.

Free Memory:

Free memory of the system in Kbytes.

Free Page Recs:

Number of page reclaims from the free list per second. The sum of page reclaims from the free list and the dirty list equals the total reclaims.

FS Blk Reads:

Number of actual disk reads initiated by the file system.

FS Blk Writes:

Number of actual disk writes initiated by the file system.

FS Read Hit:

Percentage of file system read requests that were satisfied from the buffer cache.

FS Write Hit:

Percentage of file system write requests that did not result in an actual disk write.

Interrupts:

Number of interrupts per second.

Locks Used:

Number of record/file locks currently in use.

Message Ops:

Number of System V message queue operations per second.

Number of Procs:

Number of processes in the system that are occupying process table entries.

Page Faults:

Number of page faults per second.

Page Ins:

Number of page ins (major faults) per second.

Page Outs:

Number of page outs per second.

Pages Paged In:

Number of pages paged in per second.

Pages Paged Out:

Number of pages paged out per second.

Pages Swapped In:

Number of pages swapped in per second.

Pages Swapped Out:

Number of pages swapped out per second.

Percent Locks Used:

The percentage of available record/file locks currently in use.

Procs FS I/O Wait:

Number of processes waiting for file system I/O.

Procs Phy io Wait:

Number of processes waiting for raw disk I/O.

Raw Reads:

Number of raw device reads from all devices.

Raw Writes:

Number of raw device writes from all devices.

Raw Read KB:

Number of Kbytes (KB) transferred via raw device reads.

Raw Write KB:

Number of Kbytes (KB) transferred via raw device writes.

Runnable Procs:

Number of processes that are runnable and waiting on the run queue to be dispatched to a processor.

Running Procs:

Number of processes currently running on a processor.

Semaphore Ops:

Number of System V semaphore operations per second.

Sleeping Procs:

Number of processes that are sleeping.

Swap Ins:

Number of processes swapped in per second.

Swap Outs:

Number of processes swapped out per second.

Swapped Procs:

Number of processes that are swapped to disk.

System Calls:

Number of system calls per second.

Total System Time:

Total time the system spent in system mode expressed as percent of one processor.

Total User Time:

Total time the system spent in user mode expressed as percent of one processor.

Total Time:

Sum of user time and system time expressed as percent of one processor.

Total Virtual Mem:

Sum of virtual memory sizes of all processes.

Traps:

Number of traps per second.

TTY Chars In:

Number of characters received on TTY lines per second.

TTY Chars Out:

Number of characters transmitted on TTY lines per second.

Vforks:

Number of vforks per second.

3.8 Help Window

The help window pops up whenever the user presses the help button as shown in Figure 14. This help window provides general help about the **xmon** program.

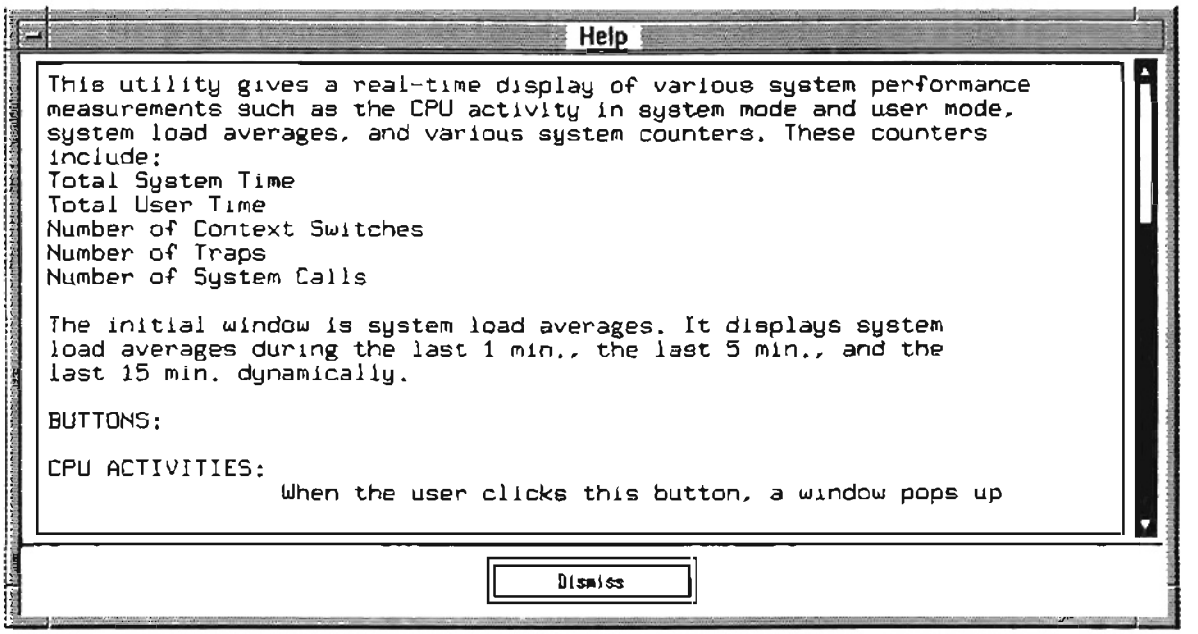


Figure 14. Help Window

3.9 Child1 Process

To capture the output of the **monitor** program, the main program forks two children: child1 and child2. The child1 process capture the activities of the processes by executing the **monitor** command and takes its output via a pipe. Because **monitor** command use curses to display its data and its output contain screen formatting as well as cursor control characters and actual data is sandwich between them, its output cannot be

used directly in the **xmon** program. The actual data needs to be extracted from its output and passed to the main program. This job is done by the child1 process.

The child1 process opens a pipe, executes the **monitor** command, and send its output through the pipe. To do this job, the following statement is executed:

file_ptr = **popen** ("**monitor**" , "r")

The **popen** command is the UNIX function call which creates a pipe between the calling program and the command to be executed [Stevens 90]. The command consists of a shell command line which is the **monitor** command in the above statement. The value returned is an I/O stream pointer that is used to read the contents of the file.

To extract the actual data from the output of **monitor** command, an algorithm was developed to track the cursor position on the screen. That algorithm uses the number of different cursor control characters that include:

^[[H: This control character puts the cursor at the top line of the screen. This character is generated whenever the "Home" key is pressed on the keyboard.

^H: This control character puts the cursor on the next line right below its current line and advances the cursor one step backward. This character is generated whenever the "Enter" key is pressed on the keyboard.

^[[nC: This control character advances the cursor n steps forward. This character is generated whenever the right cursor key is pressed on the keyboard.

^[[nD: This control character advances the cursor n steps backward. This character is generated whenever the left cursor key is pressed on the keyboard.

^[[nA: This control character advances the cursor n screen lines above its current line. This character is generated whenever the up cursor key is pressed on the keyboard.

^[[nB: This control character advances the cursor n screen lines below its current line. This character is generated whenever the down cursor key is pressed on the keyboard.

By using these control characters, we can get actual values of the CPU activities.

3.10 Child2 Process

This child captures the system counters data output by the **monitor** command. To capture the system counters, the child2 process executes the **monitor** command as:

file_ptr = **popen** ("**monitor** -f < f" , "r")

This statement executes the UNIX function call **popen** which in turn executes the **monitor** command and redirects its output to a file pointed by the pointer file_ptr.

If we execute the **monitor** command by using the -f option, we can toggle between the CPU activities screen and the system counters screen by pressing the 'f' key. The file named "f" in the above statement has a character 'f' for toggling the **monitor** screen. This is necessary to get the system counters screen of the **monitor** command.

3.11 Child3 Process

This process gets the system's load averages by executing the **uptime** command as the background process. The statement to execute the **uptime** command is as follows:

file_ptr = **popen** ("**uptime**" , "r")

Because we need to display the system's load averages in real-time, the above statement should be executed in a loop. A sample output of the **uptime** command is shown below:

03:11, up 13 days, 18:28, load average: 0.49, 0.44, 0.33

The entire output line is read as a string and only the last three values are extracted and then these values are passed to the main program via a pipe.

3.12 Programming Language

The **xmon** program was written in the C language. It has about 7783 lines of undocumented code and about 1357 lines of documentation. Its main() has about 803 lines of undocumented code. The total number of procedures of **xmon** program are 53 and the X Window routines used in this program are 18 including the routines of Xlib, Xtoolkit, and Motif.

CHAPTER IV

SUMMARY AND FUTURE WORK

4.1 Summary

The main purpose of this thesis work was to provide an interactive and user friendly interface to display dynamically the various system counters as well as the activities of the 24 CPU's of the Sequent S/81.

In Chapter I, the importance of system performance tools 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, UNIX kernel, Curses, system performance commands, the definition and importance of Graphical User Interfaces, X Window system (including its brief history, fundamental components, and software layers), and the OSF/Motif programming (including its architecture, programming structure, data types, and widget set). Chapter III described the implementation part of this thesis.

4.2 Future Work

The work for future exploration includes the following: (1) Including an algorithm to predict future system load based on the system's load history, (2) Enhancement of the

xmon program so that it can run on any machine, (3) Using the Java language to monitor network activities, and (4) Improving the algorithm of the **xmon** program so that its interaction with the user could get faster.

REFERENCES

[Ali and Yang 94] Mahir S. Ali and Cui-Qing Yang, *Xlib By Example*, AP Professional, Cambridge, MA, 1994.

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

[Chandrashekar et al. 91] S. Chandrashekar, B. E Mayfield, and Mansur H. Samadzadeh, "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, MO, pp. 119-122, April 1991.

[Chandrashekar et al. 93] S. Chandrashekar, B. E. Mayfield, and Mansur H. Samadzadeh, "Towards Automating Software Project Management," *The International Journal of Project Management*, Vol. 11, No. 1, pp. 29-38, February 1993.

[DEC 94] *DEC OSF/1 System Tuning and Performance Management*, Digital Equipment Corporation, Maynard, MA, 1994.

[Du 93] Haibo Du, *Display of Sequent Symmetry S/81 Performance Using X Window System Facilities*, MS Thesis, Computer Science Department, Oklahoma State University, Stillwater, OK, 1993.

[Flanagan 94] David Flanagan, *Motif Tools: Streamlined GUI Design and Programming with the Xmt Library*, O'Reilly & Associates, Inc., Sebastopol, CA, 1994.

[Frisch 91] Aeleen Frisch, *Essential System Administration*, O'Reilly & Associates, Inc., Sebastopol, CA, 1991.

[George and Riches 94] Alistair George and Mark Riches, *Advanced Motif Programming Techniques*, Prentice Hall International Ltd., Hertfordshire, UK, 1994.

[Heller 91] Dan Heller, *Motif Programming Manual Vol. 6*, O'Reilly & Associates, Inc., Sebastopol, CA, 1991.

[Loukides 90] Mike Loukides, *System Performance Tuning*, O'Reilly & Associates, Inc., Sebastopol, CA, 1990.

[Majidimehr 96] Amir H. Majidimehr, *Optimizing UNIX for Performance*, Prentice Hall PTR Inc., Englewood Cliffs, NJ, 1996.

[Miscovich and Simon 94] Gina Miscovich and David Simon, *The SCO Performance Tuning Handbook*, PTR Prentice Hall, Englewood Cliffs, NJ, 1994.

[Nye and O'Reilly 90] Adrian Nye and Tim O'Reilly, *X Toolkit Intrinsics Programming Manual Vol. 4*, O'Reilly & Associates, Inc., Sebastopol, CA, 1990.

[OSF 91a] Open Software Foundation, *OSF/Motif Programmer's Guide*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1991.

[OSF 91b] Open Software Foundation, *OSF/Motif Style Guide*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1991.

[Rochkind 88] M. J. Rochkind, *Advanced C Programming for Display*, Prentice Hall International Ltd., Hertfordshire, UK, 1988.

[Schreiner 90] Axel T. Schreiner, *Using C with Curses, Lex and Yacc*, Prentice Hall International Ltd., Hertfordshire, UK, 1990.

[Sebern 94] Mark J. Sebern, *Building OSF/Motif Applications: A Practical Introduction*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1994.

[Smith 94] Jerry D. Smith, *X: A Guide for User*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1994.

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

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

[Silberschatz and Galvin 94] A. Silberschatz and P. Galvin, *Operating System Concepts*, Fourth Edition, Addison-Wesley Publishing Company, Reading, MA, 1994.

[Stevens 90] W. Richard Stevens, *UNIX Network Programming*, Prentice Hall PTR, Inc., Englewood Cliffs, NJ, 1990.

[Strang 91] John Strang, *Programming with Curses*, O'Reilly & Associates, Inc., Sebastopol, CA, 1991.

[SunSoft 93] *Solaris 2.3 Administering Security, Performance, and Accounting*, Sun Microsystems, Inc., Mountain View, CA, 1993.

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

[Thobani and Samadzadeh 92] A. A. Thobani and Mansur H. Samadzadeh, "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, GA, October 1992.

APPENDICES

APPENDIX A

GLOSSARY AND TRADEMARK INFORMATION

GLOSSARY

Argument List:	An argument list is used to set various parameters of a widget such as height, width, background color, and foreground color. These values are hardcoded in the widget. It consists of an array of Arg structures, each consisting of a resource name and the value to which it should be set.
BSD:	Berkeley Standard Distribution.
Callback:	A callback is an application routine registered with a widget by the application using either the call XtAddCallback or through an argument list. A widget declares one or more callbacks as resources and an application add these callbacks in order to link widgets to the application code.
Client:	An application program connects to the window system server by an interprocess communication (IPC) path, such as a TCP connection or a shared memory buffer. This program is referred to as a client of the window system server.
Connection:	The communication path between the server and the client is known as a connection. A client usually has only one connection to the server over which requests and events are sent.
Cursor:	A cursor is the visible shape of the pointer on the screen and can be control by a mouse or the cursor keys on the keyboard.

Event:	In X Windows, an event is a data structure sent by the server describing something that just happened which may be of interest to the application. There are two major types of events: user events and window system events. The user pressing a keyboard key or clicking a mouse button generates an event, a window being moved on the screen also generates an event. It is the server's job to distribute the events to the various windows on the screen.
Graphical User Interface :	A visual representation of a program's functions.
GUI:	Graphical User Interface.
Interface:	The layer of displayed information (textually, graphically, etc.) between a user and a computer system.
Internal Fragmentation:	Amount of memory that is not being used because the allocated memory partition may be slightly larger than the requested memory.
Intrinsics:	The X Toolkit defines functions and data types that are called Intrinsics.
Load Average:	The number of active processes in a system's run queue at any time.
Mapping:	A window is said to be mapped if an XMapWindow or XMapRaised call has been performed on it. The Unmapped windows are never viewable. Mapping makes a window eligible for display.
Parent Window:	Each new window is created with reference to another previously created window. The new window is referred to as the child and the reference window is known as the parent window.
Popup:	A popup is a window that is outside the normal parental hierarchy of the geometry management. It is actually a child window of the root window that is popped up temporarily to give a user a piece of information or to get some information from the user.

Priority:	A number that determines how often the kernel will run a process. A higher priority process will run more often and therefore will finish faster than a low priority process.
Process:	A single stream of instructions.
Root Window:	Each screen has a root window covering it. It cannot be unmapped. A root window has no parent.
Round-trip Request:	A request to the server that generates a reply.
Server:	The server provides the basic windowing mechanism. It accepts the request from the client and processes it. It controls a single keyboard and pointer and one or more screen that make up a single display.
TCP/IP:	Transmission Control Protocol (TCP) and Internet Protocol (IP), the two primary components in a set of protocols that permit data transfer between a server and its clients on the same or on different networks.
X Window System:	The hardware independent and network transparent base layer that provides services to graphical user interfaces.

TRADEMARK INFORMATION

Sequent Symmetry:	A registered trademark of Sequent Computer Systems, Inc.
UNIX:	A registered trademark of AT&T Laboratories.
X Window System:	A trademark of the Massachusetts Institute of Technology.

APPENDIX B

USER MANUAL FOR **xmon**

1. Description

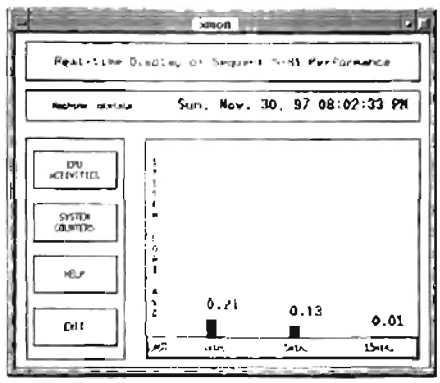
The **xmon** program is an interactive graphical user interface tool that can be used to monitor the DYNIX/ptx system's activities in real-time. This program is based on OSF/Motif and it can display various system measurements on the Sequent S/81 running DYNIX/ptx operating system. The **xmon** program use the **monitor** utility, which is the Sequent's implementation of system performance measurements, to display system activity reports.

The **xmon** program consists of three main windows: the initial window titled Real-time Graphical Display of System Measurements, the CPU activities window titled CPU Activities, and the system counters window titled System Counters. In addition to these windows, there is a help window which pops up whenever the user clicks on the help button.

The initial or main window has four buttons labeled CPU Activities, System Counters, Help, and Exit. These buttons control all the operations of **xmon**. The context sensitive help is also included to describe the various system counters.

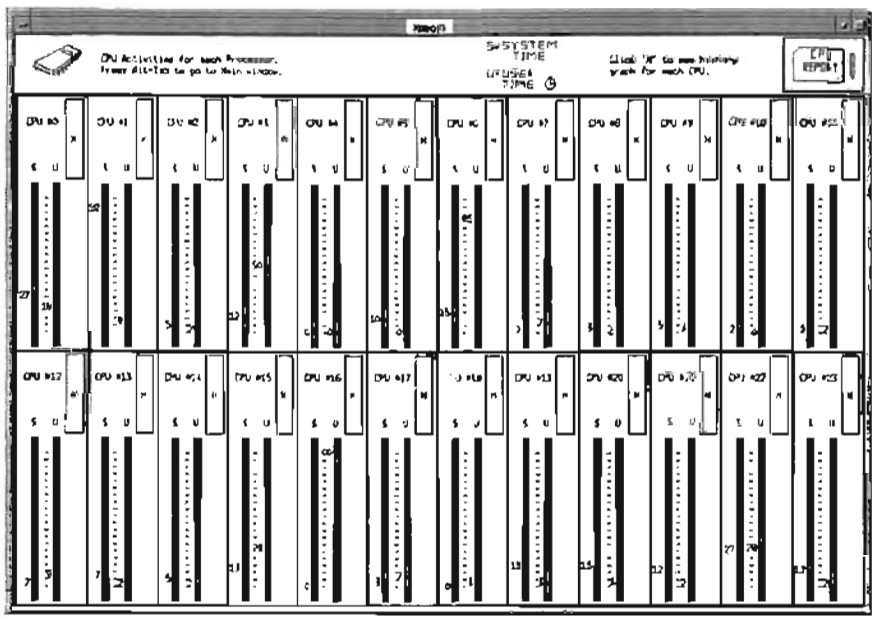
2. User Interface Objects of the **xmon** Program.

When a user invokes the **xmon** program, it shows the system load averages using a dynamic bar graph as shown below:

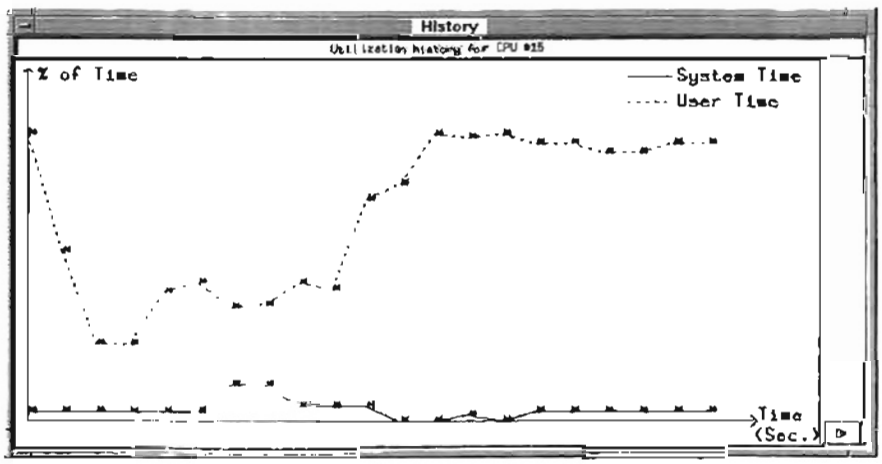


The above window shows three load averages sampled during the last one minute, the last five minutes, and the last fifteen minutes. The current date and time is displayed at the top of the window.

When the CPU activities button in the main window is pressed, a CPU activities window pops up as shown below:

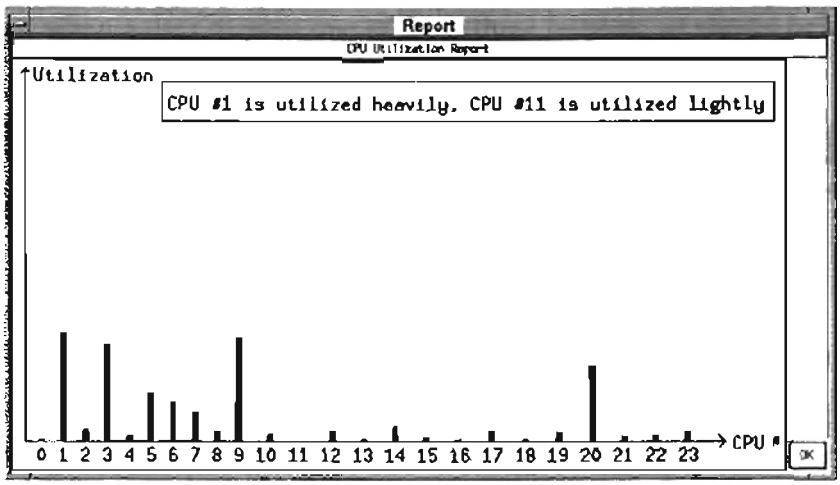


Through this CPU activities window, the user can observe the activities of the 24 CPU's on the Sequent S/81. In this window, 'S' represents system time and 'U' represents user time. Each CPU activity box has a history button labeled 'H'. When the user press this button, the history of the selected CPU is displayed in terms of the system and the user time using line graphics as show below:



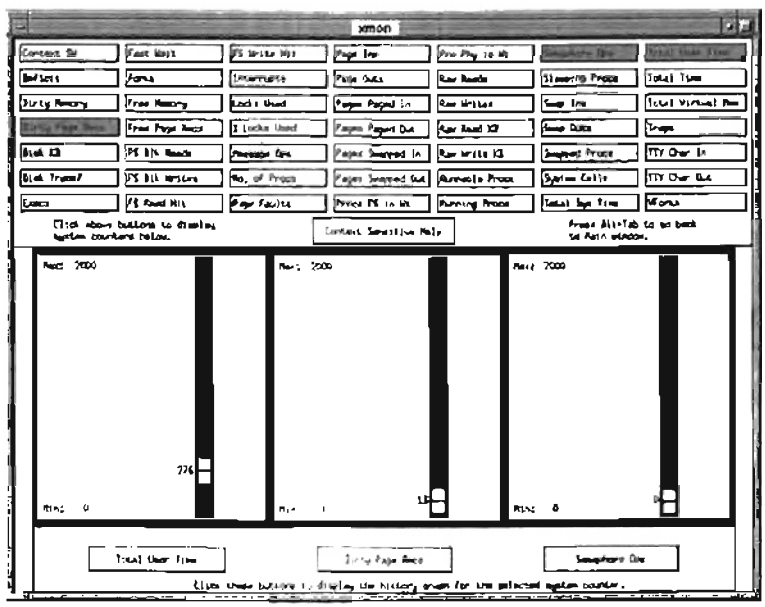
The system time is displayed by using red solid line and the user time by dashed green line. Clicking the "OK" button in the above window pops down this window.

Pressing the "CPU Report" button in the CPU activities window causes a CPU report window to pop up which shows the relative CPU utilization of the 24 microprocessors on the Sequent S/81 machine using bar graphs, as shown below:



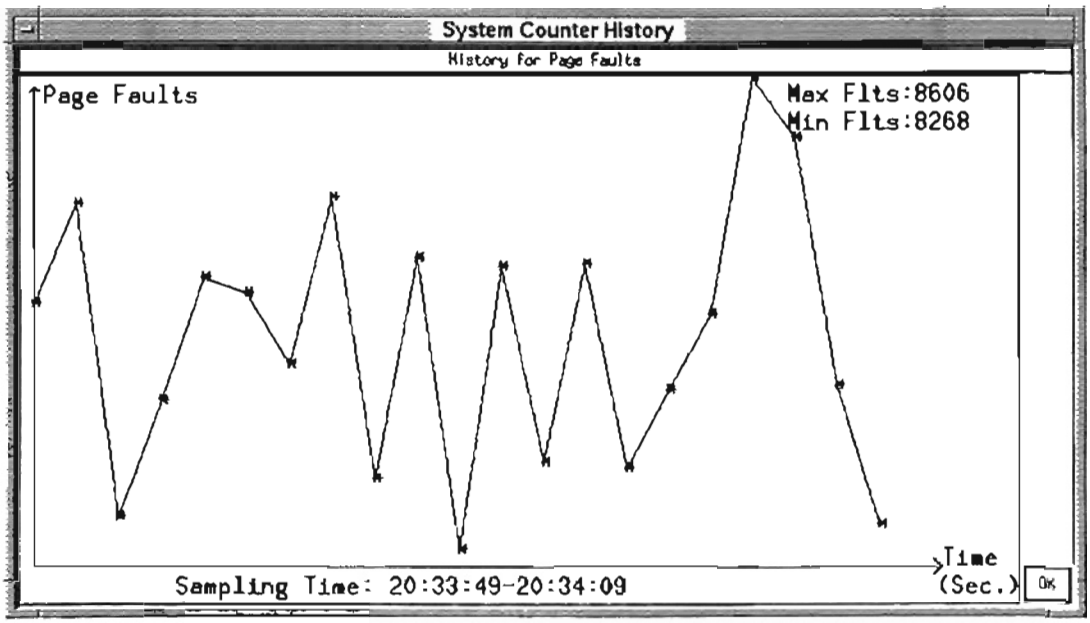
The above window also tells the user which processor is heavily used and which one is lightly used.

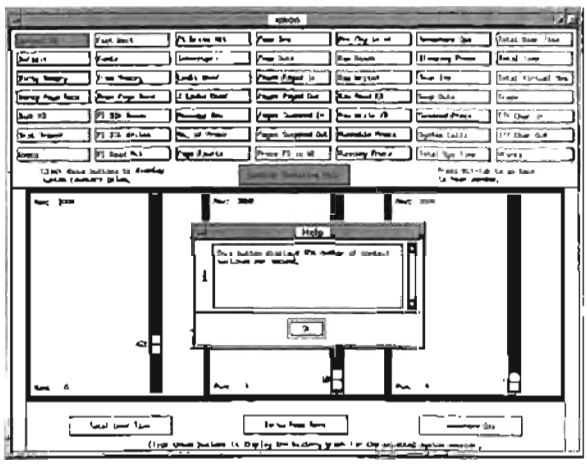
When the user press the button labeled "System Counters" in the main window, a system counter window pops up as shown below:



This window has 50 buttons that are used to select the system counters to be displayed. By default, the above window displays three system counters: Total User Time, Dirty Page Rec, and Semaphore Ops. Whenever the user select a system counter by clicking one of the buttons, the selected system counter appears on the left of the lower part of the screen.

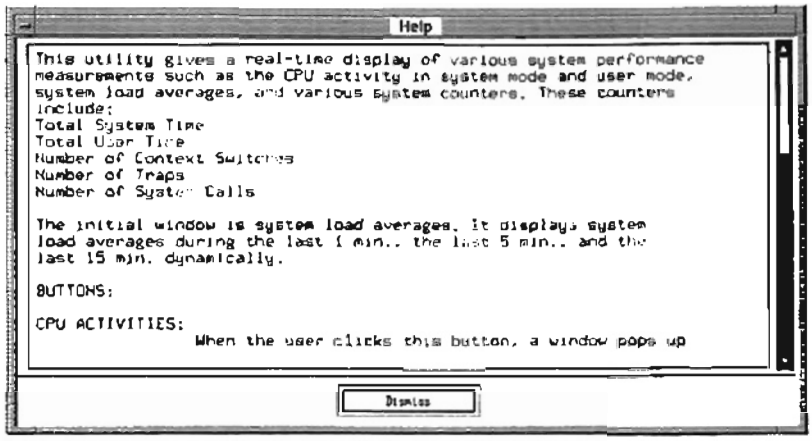
The three buttons located at the bottom of this window have two purposes: (1) to display the name of selected counters, and (2) to display the history of system counters using line graphics when a user presses one of these three buttons, as shown below:





The user may get help on individual system counters by using the context sensitive help. Whenever the user clicks on the context sensitive help button, the cursor changes its shape to a "hand" cursor and, in this condition, when the user presses any system's counter button, a help window pops up displaying the help text about the selected system counter in a small window as shown in the figure on the left.

The "Help" button in the main window can be used to display general help about the **xmon** program as shown below:



3. Related Commands

The commands related to the **xmon** program under DYNIX/ptx include **monitor**, **uptime**, **xload**, **sar**, and **ps**.

APPENDIX C

PROGRAM LISTING

The xmon program is composed of the following files:

```
context_help.txt
counter.h
help_include
help_routine
xmon.h
xmon.c
button_header
button_callbacks
store_counter
counter_window
storeit
cpu_window
```

```
/*****************************************************************/
/*                          File:  context_help.txt              */
/*****************************************************************/

/* This file contains the text for context sensitive help window. */
/* Strings for context sensitive help window. */
String context_help1[] = {            /* Context switch. */
    "This button displays the number of context switches per",
    "second.",
    "\n",
    NULL
};
String context_help2[] = {            /* Deficit. */
    "This button displays the virtual memory deficit in",
    "kbytes.",
    "\n",
    NULL
};
String context_help3[] = {            /* Dirty Memory. */
    "This button displays the dirty memory in kbytes.",
    "\n",
    NULL
};
String context_help4[] = {            /* Dirty page Recs. */
    "This button displays the number of page reclaims from",
    "the dirty list per second. The sum of page reclaims",
    "from the dirty list and the free list equals the ",
    "total reclaims.",
    "\n",
    NULL
};
String context_help5[] = {            /* Disk Kbytes. */
    "This button displays the number of kilobytes transferred",
    "per second.",
    "\n",
    NULL
};
String context_help6[] = {            /* Disk Transfer. */
    "This button displays the number of disk transfers per",
    "second.",
    "\n",
    NULL
};
String context_help7[] = {            /* Execs. */
    "This button displays the number of execs calls per",
    "second.",
    "\n",
    NULL
};
String context_help8[] = {            /* Fast wait. */
    "This button displays the number of processes that are",
    "waiting for disk I/O, page I/O, or a kernel resource.",
    "\n",
    NULL
};
String context_help9[] = {            /* Forks. */
    "This button displays the number of forks per second.",
    "\n",
    NULL
};
String context_help10[] = {           /* Free Memory. */
    "This button displays the free memory in kbytes.",
    "\n",
    NULL
};
String context_help11[] = {           /* Free page Recs. */
    "This button displays the number of page reclaims from",
    "the free list per second. The sum of page reclaims",
    "from the free list and the dirty list equals the ",
    "total reclaims.",
    "\n",
    NULL
};
String context_help12[] = {           /* FS Blk Reads. */
    "This button displays the number of actual disk reads",
    "initiated by the file system.",
    "\n",
    NULL
};
String context_help13[] = {           /* FS Blk Writes. */
```

```
    "This button displays the number of actual disk writes",
    "initiated by the file system.",
    "\n",
    NULL
};
String context_help14[] = {          /* FS Read Hit. */
    "This button displays the percentage of file system",
    "read requests that were satisfied from the buffer",
    "cache.",
    "\n",
    NULL
};
String context_help15[] = {          /* FS Write Hit. */
    "This button displays the percentage of file system",
    "write request that did not result in an actual disk",
    "write.",
    "\n",
    NULL
};
String context_help16[] = {          /* Interrupts. */
    "This button displays the number of interrupts per",
    "second.",
    "\n",
    NULL
};
String context_help17[] = {          /* Locks Used. */
    "This button displays the number of record or file",
    "locks currently in use.",
    "\n",
    NULL
};
String context_help18[] = {          /* Percent Locks Used. */
    "This button displays the percentage of available",
    "record or file locks currently in use.",
    "\n",
    NULL
};
String context_help19[] = {          /* Message Operations. */
    "This button displays the number of System V message",
    "queue operations per second.",
    "\n",
    NULL
};
String context_help20[] = {          /* No. of Processes. */
    "This button displays the number of processes in the",
    "system that are occupied process table entries.",
    "\n",
    NULL
};
String context_help21[] = {          /* Page Faults. */
    "This button displays the number of page faults per",
    "second.",
    "\n",
    NULL
};
String context_help22[] = {          /* Page ins. */
    "This button displays the number of page ins (major faults)",
    "per second.",
    "\n",
    NULL
};
String context_help23[] = {          /* Page Outs. */
    "This button displays the number of page outs per",
    "second.",
    "\n",
    NULL
};
String context_help24[] = {          /* Pages Paged In. */
    "This button displays the number of pages paged in per",
    "second.",
    "\n",
    NULL
};
String context_help25[] = {          /* Pages Paged Out. */
    "This button displays the number of pages paged out",
    "per second.",
    "\n",
    NULL
};
String context_help26[] = {          /* Pages Swapped In. */
```

```
    "This button displays the number of pages swapped in",
    "per second.",
    "\n",
    NULL
};
String context_help27[] = {           /* Pages Swapped Out. */
    "This button displays the number of pages swapped ",
    "out per second.",
    "\n",
    NULL
};
String context_help28[] = {           /* Process FS i/o wait. */
    "This button displays the number of processes that are",
    "waiting for file system I/O.",
    "\n",
    NULL
};
String context_help29[] = {           /* Process Phy io wait. */
    "This button displays the number of processes that are",
    "waiting for raw disk I/O.",
    "\n",
    NULL
};
String context_help30[] = {           /* Raw Reads. */
    "This button displays the number of raw device reads",
    "from all devices.",
    "\n",
    NULL
};
String context_help31[] = {           /* Raw Writes. */
    "This button displays the number of raw device writes",
    "from all devices.",
    "\n",
    NULL
};
String context_help32[] = {           /* Raw Read KB. */
    "This button displays the number of kilobytes (KB)",
    "transferred via raw device reads.",
    "\n",
    NULL
};
String context_help33[] = {           /* Raw Write KB. */
    "This button displays the number of kilobytes (KB)",
    "transferred via raw device writes.",
    "\n",
    NULL
};
String context_help34[] = {           /* Runnable Processes. */
    "This button displays the number of processes that are",
    "runnable and waiting on the run queue to be dispatched",
    "to a processor",
    "\n",
    NULL
};
String context_help35[]   {           /* Running Processes.  */
    "This button displays the number of processes currently",
    "running on a processor.",
    "\n",
    NULL
};
String context_help36[] = {           /* Semaphore Operations. */
    "This button displays the number of System V semaphore",
    "operations per second.",
    "\n",
    NULL
};
String context_help37[] = {           /* Sleeping process. */
    "This button displays the number of processes that are",
    "sleeping.",
    "\n",
    NULL
};
String context_help38[] = {           /* Swap Ins. */
    "This button displays the number of processes swapped",
    "in per second.",
    "\n",
    NULL
};
String context_help39[] = {           /* Swap Outs. */
    "This button displays the number of processes swapped",
```

```
    "out per second.",
    "\n",
    NULL
};
String context_help40[] = {          /* Swapped Processes. */
    "This button displays the number of processes that are",
    "swapped to disk.",
    "\n",
    NULL
};
String context_help41[] = {          /* System calls. */
    "This button displays the number of system calls per",
    "second.",
    "\n",
    NULL
};
String context_help42[] = {          /* Total Sys Time. */
    "This button displays the total time the system spent in",
    "system mode expressed as percent of on processor.",
    "\n",
    NULL
};
String context_help43[] = {          /* Total User Time. */
    "This button displays the total time the system spent in",
    "user mode expressed as percent of on processor.",
    "\n",
    NULL
};
String context_help44[] = {          /* Total Time. */
    "This button displays the sum of user and system time",
    "expressed as percent of on processor.",
    "\n",
    NULL
};
String context_help45[] = {          /* Total Virtual Memory. */
    "This button displays the sum of virtual sizes of",
    "all the processes in the system.",
    "\n",
    NULL
};
String context_help46[] = {          /* Traps. */
    "This button displays the number of traps per",
    "second.",
    "\n",
    NULL
};
String context_help47[] = {          /* TTY chars in. */
    "This button displays the number of characters received",
    "on TTY lines per second.",
    "\n",
    NULL
};
String context_help48[] = {          /* TTY chars out. */
    "This button displays the number of characters ",
    "transmitted on TTY lines per second.",
    "\n",
    NULL
};
String context_help49[] = {          /* Vforks. */
    "This button displays the number of Vforks per second.",
    "\n",
    NULL
};

                                     /* Convert all text strings into one array string so
                                     that we can access the context sensitive help text
                                     by using the index number of this string. */
String *help_texts2[] = { context_help1, context_help2, context_help3, context_help4,
context_help5, context_help6, context_help7, context_help8, context_help9, context_help10,
context_help11, context_help12, context_help13, context_help14, context_help15,
context_help16,context_help17,context_help18,context_help19,
context_help20,context_help21,context_help22,context_help23,
context_help24,context_help25,context_help26,context_help27,
context_help28,context_help29,context_help30,context_help31,
context_help32,context_help33,context_help34,context_help35,
context_help36,context_help37,context_help38,context_help39,
context_help40,context_help41,context_help42,context_help43,
context_help44,context_help45,context_help46,context_help47,
context_help48,context_help49,
};
```

```
/*****************************************************************/
/*                          File:  counter.h                     */
/*****************************************************************/

                                    /* This file contains declaration of variables and
                                    data structures for system counters window. */

#include "button_header"            /* Include file for prototypes of call back
                                    routines. */
#include "help_routine"             /* Include file for context sensitive help window.
                                    */
#include "store_counter"            /* Include the file for storing the system counters
                                    values to draw the history graphics. */

                                    /* Control characters for capturing the system
                                    counters data. */
#define H       72
#define TRUE    1
#define CELL_CTR    76              /* No. of cells on screen */

int pipe_cntrA[2];                  /* Pipes for communication between the parent and
                                    child process. */
int pipe_cntrB[2];
int pipe_getdata[2];

                                    /* Index variables for the array of data structure
                                    for graph window. */
int idn0,idn1,idn2,idn3,idn4,idn5,idn6,idn7,idn8,idn9,idn10;
int idn11,idn12,idn13,idn14,idn15,idn16,idn17,idn18,idn19,idn20;
int idn21,idn22,idn23;

int Hflag=0;
int input_select=0;                 /* Selected screen line. */
int selected_line=2;
Display *dpy_ctr;                   /* Display ID. */
Widget scale_ct[8],toplevel_ctr;
Widget label1,label2,label3;
Widget disp_bt1,disp_bt2,disp_bt3;
Widget lb_m1,lb_m2,lb_m3;           /* Label for maximum & minimum values. */
Display *dpy_popup_CTR;
Screen *scr_popup_ctr;              /* Screen ID. */
int ct_fg0=0;
int ct_fg1=0;
int ct_fg2=0;
int which_ctr=0;
Colormap cmap;                      /* X Window color information. */
long tim_v;
struct tm *today_ctr;               /* Date and time data structure. */

                                    /* Data structure for history graph. */
typedef struct{
        int first_pval;             /* First screen value. */
        int v2nd_pval;              /* 2nd screen value. */
        int v3rd_pval;              /* 3rdt screen value. */
        }graph_values;
        graph_values prm[25];


                                    /* Data structure for three push buttons located at
                                    the bottom of the system counter window. */
typedef struct{
        char Label[22];             /* Label of buttons. */
        int Line;                   /* Line number of buttons. */
        int Val_pos;                /* Values of screen position. */
        }BT_LB;
        BT_LB Counter[3];           /* For three push buttons. */

                                    /* Data structure to store current system counter
                                    values. */
typedef struct{
        int VFirst[24];             /* First screen value. */
        int V2nd[24];               /* 2nd screen value. */
        int V3rd[24];               /* 3rdt screen value. */
        }VC;
        VC vc[24];


                                    /* Data structure to store the previous system
                                    counter values. */
typedef struct{
```

```
        int PFirst;                     /* First previous screen value. */
        int P2nd;                       /* 2nd previous screen value. */
        int P3rd;                       /*3rd previous screen value. */
        }PV;
        PV Prev[24];

                                        /* Data structure to simulate the actual screen. */
typedef struct{
        int sl[CELL_CTR];               /* Screen line. */
        }scrst;
        scrst acct[24];                 /* Screen with 24 rows. */

                                        /* Data structure to hold the last character of the
                                        system counters. */
typedef struct{
        char lastch[2];                 /* Last character of counters */
        int ary[25];
        int skp[24];                    /* Cursor skip values. */
        int cur_add[24];                /* Cursor addition values. */
        }last;
last wrd[47];

int cur_pos_ctr=1;                      /* Default cursor position. */
int line_ctr=0;                         /* Default screen line number. */
int first_time=1;

                                        /* Prototypes of routines to capture the system
                                        counter data. */
int convt_ctr(int j,int k);
void f_item(FILE *fp,char *item,int y,int n);
void fn2(int i,char *item);
void fn3(int i,char *item);
int get_values(char *item,int *i);
void put_line(int data);
void set_cursor(int ct_char,int tab,int val);
void initialization();
void send_data();
int convert_data(int a,int b,int c,int d,int e,int f);

                                        /* Prototypes for the routines used in system
                                        counter window. */
void show_graph();
void child1(void);
void initialize_graph();
void max_min(int *max,int *min,int kd_line,int val,int pos);
void show_utilization();
void Popdown();
void show_stat();
void store_it(int line,int V1st,int V2nd);
void Sinput();
void y_name(char *,int,int);
void Popdown2();
void
show_utilization2(Widget w, XtPointer data, XmDrawingAreaCallbackStruct *cbk);


  /*****************************************************************/
 /*                          File:  help_include                    */
/*****************************************************************/

                                        /* This file contains the help routine and help text
                                        for main window. */

#include <ctype.h>
#include <Xm/DialogS.h>                 /* Include routines for dialog shell. */
#include <Xm/Text.h>                    /* Include routines for text widget. */
#include <Xm/PushBG.h>                  /* Include routines for push buttons. */
#include <Xm/PanedW.h>                  /* Include routines for pane window. */
#define  BUFF  3024                     /* Buffer size for help text. */

                                        /* Help routine to display the help window. */
void help_cb(Widget w, XtPointer client_data, XtPointer call_data);
                                        /* Routine to get the parent of the widget. */
Widget GetTopShell(Widget w);
                                        /* Bitmap for the  help icon. */
char cur_bitmap3[45]="/z/rsyedna/motif/part1/help_icon";

                                        /* String to hold help text. */
String context_help[] = {
```

```
    "This utility gives a real-time display of various system",
    "performance measurements such as CPU activity in system",
    "mode and user mode, system load averages, and various system",
    "counters. These counters include:\n",
    "Total System Time\n",
    "Total User Time\n",
    "Number of Context Switches\n",
     Number of Traps\n",
    "Number of System Calls\n",
    "\n",
    "The first display on the initial window is the system load",
    "averages.\nIt displays system load averages during the last ",
    "1 min., the last 5 min., and the last 15 min. dynamically.\n ",
    "\n",
    "BUTTONS:\n",
    "\n",
    "CPU ACTIVITIES:\n",
    "           When the user clicks this button, a window pops up\n",
    "           and presents the activities of all on-line 24 CPUs\n",
    "           of Sequent S/81 System in real-time. These activities\n",
    "           are divided into system time and user time.\n",
    "           This window has a push button labeled CPU Report.\n",
    "           Whenever the user presses this button, a window pops up\n",
    "           and shows a CPU utilization report about all the 24 \n",
    "           microprocessors using bar graphs. The push button \n",
    "           labeled 'H' display the history of each CPU over\n",
    "           certain time period.\n",
    "\n",
    "SYSTEM COUNTERS:\n",
    "           When the user clicks this button, a window pops up\n",
    "           and presents a real-time display of various system\n",
    "           counters inside the Kernel. These counters are\n",
    "           incremented or decrement as various system actions\n",
    "           occur. The three push buttons located at the bottom\n",
    "           of this window provide history for three selected system\n",
    "           counters. The context sensitive help button is used to \n",
    "           provide help for each push button for system counters.\n",
    "           To use it, first click on the context sensitive button \n",
    "           then click any system counter push button.\n",
    "\n",
    "HELP:\n",
    "           This button provides help to the user.\n",
    "\n",
    "EXIT:\n",
    "           This button quits the application and cleans up some\n",
    "           child processes.\n",
    "\n",
    "Author:          Syed Nasir Raza and Dr. Mansur Samadzadeh\n",
    "E-mail:          rsyedna@a.cs.okstate.edu\n",
    "           @1997\n",
    "\n",
    "NOTE:            User should expect short delay in the program's\n",
    "                 response due to the amount of calculation involved\n",
    "                 and the dynamic nature of the data.\n",
    NULL
};

String context_hp2[] = {
    "",
    NULL
};


String *help_texts[] = {
    context_help,
    context_hp2,
};

                                        /* This routine is used to get the parent of the
                                        widget. */
Widget GetTopShell(w)
Widget w;
{
        while (w && !XtIsWMShell (w))
                w = XtParent (w);       /* Get the parent. */
        return w;
}

                                        /* This routine executes as the call backs routine
                                        for the help push button on the main window. This
```

```
                                routine pops up a window and displays the help text.
                                */
void
help_cb(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        Widget help_dialog, pane, text_w, form, sep, widget, label;
        extern void DestroyShell();     /* Routine to destroy the shell. */
        Pixmap pixmap;                  /* Bitmap graphics for help icon. */
        Pixel fg, bg;                   /* Foreground and background pixels. */
        Arg args[10];
        int n = 0;
        int i;
        char *p, buf[BUFF];
        Dimension h;                    /* Position of window on the screen. */
        int item_no;
        Cursor cursor;
        XSetWindowAttributes attrs;
        extern Display *dpy_main;
        extern Widget toplevel;

                                        /* Change the cursor to busy state. */
        cursor=XCreateFontCursor(dpy_main,XC_watch);
        attrs.cursor=cursor;
        XChangeWindowAttributes(dpy_main,XtWindow(toplevel),CWCursor,&attrs);


                                        /* Create the pop up help dialog shell. */
        help_dialog = XtVaCreatePopupShell ("Help",
                                        /* Parent. */
        xmDialogShellWidgetClass, GetTopShell (w),
        XmNdeleteResponse, XmDESTROY,
        XmNx, 210, XmNy, 360,           /* Size of shell. */
        NULL);

                                        /* Create a Paned Window to manage the stuff in this
                                        dialog. */
        pane = XtVaCreateWidget ("pane", xmPanedWindowWidgetClass, help_dialog,
        XmNsashWidth,  1,               /* Paned Window won't let us set these to 0! */
        XmNsashHeight, 1,               /* Make small so user doesn't try to resize */
        NULL);

                                        /* Create a RowColumn in the form for Label and Text
                                        widgets. This is the control area. */
        form = XtVaCreateWidget ("form", xmFormWidgetClass, pane, NULL);
        XtVaGetValues (form,            /* once created, we can get its colors. */
        XmNforeground, &fg,
        XmNbackground, &bg,
        NULL);

                                        /* Create the pixmap of the appropriate depth using
                                        the colors that will be used by the parent (form).
                                        */
        pixmap = XmGetPixmap (XtScreen (form), cur_bitmap3, fg, bg);

                                        /* Create a label gadget using this pixmap */
        label = XtVaCreateManagedWidget ("label", xmLabelGadgetClass, form,
        XmNlabelType,         XmPIXMAP, /* Pixmap */
        XmNlabelPixmap,       pixmap,
        XmNleftAttachment,    XmATTACH_FORM,
        XmNtopAttachment,     XmATTACH_FORM,
        XmNbottomAttachment, XmATTACH_FORM, NULL);

                                        /* Prepare the text for display in the ScrolledText
                                        object we are about to create. */
        XtVaGetValues(w, XmNuserData, &item_no, NULL);
        for (p = buf, i = 0; help_texts[item_no][i]; i++) {
                p += strlen (strcpy (p, help_texts[item_no][i]));
                if (!isspace (p[-1])) /* Spaces, tabs and newlines are spaces. */
                *p++ = ' ';             /* Lines are concatenated together, insert a space.
                                        */
        }
        *--p = 0;                       /* Get rid of trailing space. */
                                        /* Set the parameters for the text widget. */
                                        /* Make window scroll vertically. */
        XtSetArg (args[n], XmNscrollVertical,         True); n++;
                                        /* Window is not allow to scroll horizontally. */
        XtSetArg (args[n], XmNscrollHorizontal,       False); n++;
```

```
                                /* Set edit mode. */
        XtSetArg (args[n], XmNeditMode,              XmMULTI_LINE_EDIT); n++;
                                /* Edit is not allowed. */
        XtSetArg (args[n], XmNeditable,              False); n++;
                                /* Make cursor invisible. */
        XtSetArg (args[n], XmNcursorPositionVisible, False); n++;
                                /* Word wrap in ON. */
        XtSetArg (args[n], XmNwordWrap,              True); n++;
                                /* Help text. */
        XtSetArg (args[n], XmNvalue,                 buf); n++;
                                /* Set number of rows. */
        XtSetArg (args[n], XmNrows,                  10); n++;
                                /* Set the width and height of help window. */
        XtSetArg (args[n], XmNwidth,                 700); n++;
        XtSetArg (args[n], XmNheight,                300); n++;

                                /* Create the text widget to display help text. */
        text_w = XmCreateScrolledText(form, "help_text",args, n);

        XtVaSetValues (XtParent (text_w),
        XmNleftAttachment,   XmATTACH_WIDGET,
        XmNleftWidget,       label,
        XmNtopAttachment,    XmATTACH_FORM,
        XmNrightAttachment,  XmATTACH_FORM,
        XmNbottomAttachment, XmATTACH_FORM,
        NULL);
        XtManageChild (text_w);
        XtManageChild (form);

                                /* Create another form to act as the action area for
                                the dialog */
        form = XtVaCreateWidget ("form2", xmFormWidgetClass, pane, XmNfractionBase, 5,
        NULL);

                                /* The Dismiss button is under the pane's separator
                                and is attached to the left edge of the form.  It
                                spreads from position 0 to 1 along the bottom (the
                                form is split into 5 separate grids via
                                XmNfractionBase upon creation). */
        widget = XtVaCreateManagedWidget ("Dismiss",
        xmPushButtonGadgetClass, form,
        XmNtopAttachment,        XmATTACH_FORM,
        XmNbottomAttachment,     XmATTACH_FORM,
        XmNleftAttachment,       XmATTACH_POSITION,
        XmNleftPosition,         2,
        XmNrightAttachment,      XmATTACH_POSITION,
        XmNrightPosition,        3,
        XmNshowAsDefault,        True,
        XmNdefaultButtonShadowThickness, 1,
        NULL);

                                /* set the call back routine for Dismiss button. */
        XtAddCallback (widget, XmNactivateCallback, DestroyShell, help_dialog);

                                /* Manage the form. */
        XtManageChild (form);
                                /* Get the height of window. */
        XtVaGetValues (widget, XmNheight, &h, NULL);
                                /* Set the maximum and minimum height. */
        XtVaSetValues (form, XmNpaneMaximum, h, XmNpaneMinimum, h, NULL);
                                /* Manage the pane widget. */
        XtManageChild (pane);
                                /* Display the help window on the screen. */
        XtPopup (help_dialog, XtGrabNone);
                                /* Set the cursor to its normal shape. */
        attrs.cursor=None;
        XChangeWindowAttributes(dpy_main,XtWindow(toplevel),CWCursor,&attrs);

}

                                /* This is call back routine for Dismiss push
                                button. It function is to free up memory occupied by
                                the help window. */
void
DestroyShell(widget, client_data, call_data)
Widget widget;
XtPointer client_data;
XtPointer call_data;
{
        Widget shell = (Widget) client_data;
```

```
        XtDestroyWidget(shell);         /* Destroy the shell and free the occupied memory.
                                        */
}


  /*****************************************************************/
 /*                          File:  help_routine                  */
/*****************************************************************/

                                        /* This file contains the routine for context
                                        sensitive help window. */
#include "context_help.txt"             /* Include the context sensitive help text file. */
                                        /* Prototype for context sensitive help routine. */
void help_cb2(Widget w);
                                        /* Bitmap for the  help icon. */
char cur_bitmap4[48]="/z/rsyedna/motif/part1/context_icon";

                                        /* This routine executes as the call backs routine
                                        for the context sensitive help push button on the
                                        system counters window. This routine pops up a
                                        window and displays the help text. */
void
help_cb2(w)
Widget w;
{
        Widget help_dialog, pane, text_w, form, sep, widget, label;
        extern void DestroyShell();     /* Routine to destroy the shell. */
        Pixmap pixmap;                  /* Bitmap graphics for help icon. */
        Pixel fg, bg;                   /* Foreground and background pixels. */
        Arg args[10];
        int n = 0;
        int i;
        char *p, buf[BUFF];
        Dimension h;                    /* Position of window on the screen. */
        int item_no;

                                        /* Create the pop up context sensitive help dialog
                                        shell. */
        help_dialog = XtVaCreatePopupShell ("Help",
        xmDialogShellWidgetClass, GetTopShell (w), /* Parent. */
        XmNdeleteResponse, XmDESTROY,
        XmNx, 210, XmNy, 360,           /* Size of shell. */
        NULL);

                                        /* Create a PanedWindow to manage the stuff in this
                                        dialog. */
        pane = XtVaCreateWidget ("pane", xmPanedWindowWidgetClass, help_dialog,
        XmNsashWidth,  1,               /* PanedWindow won't let us set these to 0! */
        XmNsashHeight, 1,               /* Make small so user doesn't try to resize */
        NULL);

                                        /* Create a RowColumn in the form for Label and Text
                                        widgets. This is the control area. */
        form = XtVaCreateWidget ("form1", xmFormWidgetClass, pane, NULL);
        XtVaGetValues (form,            /* once created, we can get its colors. */
        XmNforeground, &fg,
        XmNbackground, &bg,
        NULL);

                                        /* Create the pixmap of the appropriate depth using
                                        the colors that will be used by the parent (form).
                                        */
        pixmap = XmGetPixmap (XtScreen (form), cur_bitmap4, fg, bg);

                                        /* Create a label gadget using this pixmap */
        label = XtVaCreateManagedWidget ("label", xmLabelGadgetClass, form,
        XmNlabelType,         XmPIXMAP,
        XmNlabelPixmap,       pixmap, /* Pixmap */
        XmNleftAttachment,    XmATTACH_FORM,
        XmNtopAttachment,     XmATTACH_FORM,
        XmNbottomAttachment,  XmATTACH_FORM,
        NULL);

                                        /* Get the push button's number so that help's text
                                        is displayed related to that push button. */
        XtVaGetValues(w, XmNuserData, &item_no, NULL);
                                        /* Prepare the text for display in the ScrolledText
                                        object we are about to create. */
        for (p = buf, i = 0; help_texts2[item_no][i]; i++) {
```

```
p += strlen (strcpy (p, help_texts2[item_no][i]));
if (!isspace (p[-1]))          /* Spaces, tabs and newlines are spaces. */
*p++ = ' ';                    /* Lines are concatenated together, insert a space.
                               */
}
*--p = 0;                      /* Get rid of trailing space. */
                               /* Set the parameters for the text widget. */
                               /* Make window scroll vertically. */
XtSetArg (args[n], XmNscrollVertical,        True); n++;
                               /* Window is not allow to scroll horizontally. */
XtSetArg (args[n], XmNscrollHorizontal,      False); n++;
                               /* Set edit mode. */
XtSetArg (args[n], XmNeditMode,              XmMULTI_LINE_EDIT); n++;
                               /* Edit is not allowed. */
XtSetArg (args[n], XmNeditable,              False); n++;
                               /* Make cursor invisible. */
XtSetArg (args[n], XmNcursorPositionVisible, False); n++;
                               /* Word wrap in ON. */
XtSetArg (args[n], XmNwordWrap,              True); n++;
                               /* Help text. */
XtSetArg (args[n], XmNvalue,                 buf); n++;
                               /* Set number of rows. */
XtSetArg (args[n], XmNrows,                  6); n++;
                               /* Set the width and height of help window. */
XtSetArg (args[n], XmNwidth,                 300); n++;
XtSetArg (args[n], XmNheight,                100); n++;

                               /* Create the text widget to display context
                               sensitive help. */
text_w = XmCreateScrolledText(form, "help_text",args, n);
XtVaSetValues (XtParent (text_w),
XmNleftAttachment,   XmATTACH_WIDGET,
XmNleftWidget,       label,
XmNtopAttachment,    XmATTACH_FORM,
XmNrightAttachment,  XmATTACH_FORM,
XmNbottomAttachment, XmATTACH_FORM,
NULL);
XtManageChild (text_w);

                               /* Set the background color of context sensitive
                               help window to green. */
XtVaSetValues(text_w,XtVaTypedArg,XmNbackground, XmRString,"yellow",8,NULL);

XtManageChild (form);

                               /* Create another form to act as the action area for
                               the dialog */
form = XtVaCreateWidget ("form2", xmFormWidgetClass, pane,
XmNfractionBase, 5, NULL);

                               /* The OK button is under the pane's separator and
                               is attached to the left edge of the form.  It
                               spreads from position 0 to 1 along the bottom (the
                               form is split into 5 separate grids via
                               XmNfractionBase upon creation). */

widget = XtVaCreateManagedWidget ("OK",
xmPushButtonGadgetClass, form,
XmNtopAttachment,        XmATTACH_FORM,
XmNbottomAttachment,     XmATTACH_FORM,
XmNleftAttachment,       XmATTACH_POSITION,
XmNleftPosition,         2,
XmNrightAttachment,      XmATTACH_POSITION,
XmNrightPosition,        3,
XmNshowAsDefault,        True,
XmNdefaultButtonShadowThickness, 1,
NULL);


                               /* Set the call back routine for OK button. */
XtAddCallback (widget, XmNactivateCallback, DestroyShell, help_dialog);
                               /* Manage the form. */
XtManageChild (form);
                               /* Get the height of window. */
XtVaGetValues (widget, XmNheight, &h, NULL);
                               /* Set the maximum and minimum height. */
XtVaSetValues (form, XmNpaneMaximum, h, XmNpaneMinimum, h, NULL);
                               /* Manage the pane widget. */
XtManageChild (pane);
```

```
                                    /* Display the context sensitive help window on the
                                    screen. */
        XtPopup (help_dialog, XtGrabNone);
}


  /*****************************************************************/
 /*                          File: xmon.h                         */
/*****************************************************************/

                                    /* This file contains the declaration of variables
                                    and data structures for the main program. */

#include <stdio.h>                  /* Routines for standard input and output. */
#include <string.h>                 /* Routines for string operations. */
#include <time.h>                   /* Routines to find out the time and date. */
#include <signal.h>                 /* Routines for process signal operations. */
#include <sys/types.h>              /* Routines for fork operations. */
#include <Xm/PushB.h>               /* Motif header Routines for push button. */
#include <X11/Xlib.h>               /* X Window's basic library. */
#include <Xm/MainW.h>               /* Motif header Routines for main window. */
#include <Xm/DrawingA.h>            /* Motif header Routines for drawing area. */
#include <Xm/Label.h>               /* Motif header Routines for creating label widget.
                                    */
#include <Xm/Scale.h>               /* Motif header Routines for scale widget. */
#include <Xm/Form.h>                /* Motif header Routines for creating form. */
#include <Xm/Frame.h>               /* Motif header Routines for frame widget. */
#include <Xm/LabelG.h>              /* Motif header Routines for label gadget. */
#include <Xm/RowColumn.h>           /* Motif header Routines for row and column widget.
                                    */
#include <X11/cursorfont.h>         /* X Window header Routines for cursor shapes. */
#define MAX_VAL_MARK    20
#define MAX             1024
#define MAX_VAL         20
#define MAX_ITM         100
#include "help_include"             /* Help routine for main window. */
#include "storeit"                  /* Routines to store CPU data in data structures. */
#include "counter.h"                /* Header files for system counter window. */

XtAppContext app2;                  /* Application context for main window. */
                                    /* Widgets declaration for main window. */
Widget toplevel,time_widget;
Widget draw2;
Widget wid_cpu;
                                    /* Display IDs for main window. */
Display *display;
Display *display2;
Display *dpy_main;
                                    /* Pipes declaration for xmon program. */
int p_main[2];                      /* Pipe for main window. */
int cpu_pipe[2];                    /* Pipe for CPU window. */
int counter_pipe[2];                /* Pipe for counter window. */
int counter_pipe2[2];               /* Pipe for counter window. */
                                    /* System font for main window. */
String fallbacks[] =
{"*help_text.fontList:9x15=charset","*draw_tm.fontList:Fixed=charset",
"*title.fontList:9x15=charset", NULL };

struct tm *today;                   /* Structure to calculate current date and time of
                                    day. */
char time_buf[33];                  /* Variable to hold time. */
char time_buf2[20];                 /* Variable to hold date. */
long tim;                           /* Variable to hold time and date. */


GC gc2;                             /* Graphics Context for main window. */
XColor xcolour,spare;               /* Structure to hold color information. */
long int fill_pixel=1;
XmString str_activity1;             /* Motif string data type. */
int input_select2=0;                /* Flag to indicate that main window has been
mapped.                                 */

                                    /* Prototypes for main routine. */
                                    /* To update the information in the drawing area. */
void draw_cbk(Widget , XtPointer ,XmDrawingAreaCallbackStruct *);
                                    /* To convert ASCII to integer. */
int cvt(int v);
                                    /* To activate CPU window. */
void cpuwin_callback(Widget , XtPointer , XtPointer);
```

```
                                        /* To activate system counter window. */
void counter_callback(Widget , XtPointer , XtPointer);
                                        /* To load font. */
void load_font(XFontStruct **);
                                        /* To paint graphics on main window. */
void pnt(Widget);
                                        /* Child process for CPU window. */
void chld2(int,char **);
                                        /* Child process for system counter window. */
void child3(int,char **);
                                        /* Child process to capture data for CPU window. */
void child_cpu_act(void);
                                        /* Child process to capture system load averages. */
void child_load(void);

                                        /* Structure to store CPU data to draw history
                                        graphics. */
typedef struct{                         /* Data for system time. */
                        int sys_time[24];
                                        /* Data for user time. */
                        int usr_time[24];
                        }VPRO;
                                        /* For 24 CPU's. */
                        VPRO Vcpu[24];


                                        /* Structure for CPU utilization graphics. */
typedef struct{
                        int cpu_util;
                }util;
        util cpu_utilization[25];

                                        /* Structure for CPU representation on CPU window.
                                        */
typedef struct{
                                Widget scale_a[2];
                                }scrln;
                scrln stline[25];


Display *dpy_popup;                     /* Display ID for pop up window. */
Display *dpy;                           /* Display ID for CPU window. */
Screen *scr_ptr_popup;                  /* Screen ID for CPU window. */
Widget  frame_a[25];
XtAppContext app;                       /* Application Context. */
XEvent event;                           /* Events for CPU window. */
int p[2],pipe2[2];                      /* Pipes for CPU window. */
int pid,flag3 = 0;
Pixel fg, bg;                           /* Foreground and background color information for
                                        CPU window graphics. */
char cur_bitmap[10]="cpu_icon";         /* Bitmap graphics for CPU window. */
char cur_bitmap2[6]="scpu";             /* Bitmap graphics for CPU window. */
Pixmap pixmap;                          /* Pixel information for drawing area. */
Pixmap pixmap2;

                                        /* Prototype for CPU history graphics. */
void show_cpu_stat();
void cpu_value_store();
void show_cpu_graph();
void cpu_stat_popdown();
void report_ini();
void bcall();

                                        /* Curses control characters. */
#define A       65
#define C       67
#define D       68
#define B       66
#define H_CONT   8
#define CELL    80                      /* Number of columns on the screen. */

                                        /* Structure to hold captured data from curses
                                        screen. */
                                        /* Structure for screen line. */

                typedef struct{         /* Line #. */
                                int n;
                                        /* Screen cells. */
                                int L[CELL];
                                        /* Index into screen cells. */
```

```
                                int indx;
                               }screenline;

                screenline sline[22];/* Number of rows = 22. */


int flag=0;                             /* Flag to indicate that line is blank. */
int vflag=0;                            /* Flag to indicate that the program receive values.
                                           */
int flag_ok=0;                          /* Flag to take care of going up from line#22 first
time. */
int dflag=0;                            /* Flag to indicate that program. expects some data
                                        */
int flg_space=0;
int sp_ctr=0;                           /* Space character. */
int line=0;                             /* Initial line value. */
int cur_pos=1;                          /* Initial cursor position on the screen. */

                                        /* Prototypes for capturing CPU data from curses
                                        screen. */
void cr(FILE *fp,char *,int,int);
void fn(int i,int n,char *item);
int value(int i,int n,char *item);
int is_go_top(int i,int n,char *item);
int convt(int j,int k);
void update_sline(int s,int u,int Ln,int g);
void new_line(int control_char,int s,int u,int Ln,int val,int g);
void backward(int control_char,int s,int u,int Ln,int val,int g);
void fward(int control_char,int s,int u,int Ln,int flag,int val,int g);
void upward(int);
void downward(int);
int roundval(int x);
void ctr_char(int control_char,int sys,int user,int line,int flag,int val,int g);


  /*****************************************************************/
 /*                          File: xmon.c                         */
/*****************************************************************/

                                        /* This file contains the main program. */
#include "xmon.h"                       /* header file for data structures and global
                                        variables.
                                        */

                                        /* This is the main routine of xmon program. The
                                        jobs of this main routine is to initialize all data
                                        structures of xmon program, create three main
                                        children of xmon program that gather CPU and system
                                        counter data as well as system load averages,
                                        connects to X server, map main program window on the
                                        screen, display system load averages using bar
                                        graphics, and take user inputs. */
main(argc, argv)
int argc;
char *argv[];
{                                       /* Declear variables for main window widgets. */
Widget  button,form,form2,form3,frame,frame2,frame3;
Widget frame4,form5,exitb,draw,Help,Sys_ctr;
Widget frame_tm,form_tm,lb_load_name;
Widget last_lb[4];
XmString btn_text;                      /* Motif string type for push button label. */
XGCValues gcv;                          /* Graphics context variable. */
Window win2;                            /* Window structure for display date and time. */
Screen *screen_ptr;                     /* Screen structure for drawing area. */
Colormap cmap;                          /* X Window color structure to set the color. */
GC gc;                                  /* Graphics context drawing area. */
int pid1,pid2,pid3;                     /* Process Ids for the three children of main
                                        routine.
                                        */
int i,p_fg=0;
                                        /* Label for y-axis for system load averages. */
char lable_name[]="S\nY\nS\nT\nE\nM\n\nL\nO\nA\nD\n\nA\nV\nG";
                        /* Title string of main window of xmon program. */
XmString Title_st=
XmStringCreateSimple("Real-time Display of Sequent S/81 Performance");
Font font;                              /* X Window font structure. */
XmFontList fontlist;                    /* Font list structure to add user defined fonts. */
```

```
                                        /* Display status message when xmon program start.
                                        */
printf("\n\n\tInitializing xmon data structures.....\n");
printf("\tConecting to X Server.....\n");
printf("\tPlease Wait.....\n");

                                        /* Initialize index variables for CPU data
                                        structures.
                                        */
cind0=cind1=cind2=cind3=cind4=cind5=cind6=cind7=cind8=cind9=0;
cind10=cind11=cind12=cind13=cind14=cind15=cind16=cind17=cind18=cind19=0;
cind20=cind21=cind22=cind23=0;

                                        /* Create pipe to control  the CPU window process.
                                        */
if(pipe(cpu_pipe)<0) {printf("error in cpu_pipe\n"); exit(0);}

                                        /* Create pipe to get the system load average data
                                        from the child process.
                                        */
if(pipe(p_main)<0) {printf("error in p_main\n");exit(0);}
                                        /* Create pipe to invoke the system counter window
                                        process.
                                        */
if(pipe(counter_pipe)<0) {printf("error in counter_pipe\n");exit(0);}
                                        /* Create pipe to get the system counter data from
                                        the child process.
                                        */
if(pipe(pipe_cntrB)<0) {printf("error in pipe\n");exit(0);}
                                        /* Create pipe to control the system counter window
                                        process.
                                        */
if(pipe(pipe_getdata)<0) {printf("error in pipe\n");exit(0);}

pid1=fork();                            /* Create child process. */
if(pid1==0)
{                                       /* Child process 1. */
        child_load();                   /* Gather system load average data. */
}
else
{                                       /* Parent process. */
        pid2=fork();                    /* Create 2nd child process. */
        if(pid2==0)
        {
                child2(argc, argv);     /* Gather CPU activities data. */

        }
        else
        {                               /* Parent process. */
                pid3=fork();            /* Create 3rd child process. */
                if(pid3==0)
                {                       /* Gather system counter activities. */
                        child3(argc, argv);
                }
                else
                {                       /* Parent process. */
                                        /* Initialize X application and connect to x server.
                                        */
toplevel = XtVaAppInitialize (&app2, "Demos", NULL, 0, &argc, argv, fallbacks, NULL);
                                        /* Get the display ID of root window. */
dpy_main=XtDisplay(toplevel);

                                        /* Load new fonts and add them to font data
                                        structure. These fonts are used in help window and
                                        title label.
                                        */
font = XLoadQueryFont (dpy_main, "-ncd-terminal-medium-r-narrow--14-105-100-100-c-60-
iso8859-1");
fontlist = XmFontListCreate (font, "tag1");
font = XLoadQueryFont (dpy_main, "-ncd-terminal-medium-r-narrow--14-105-100-100-c-60-
iso8859-1");
fontlist = XmFontListAdd (fontlist, font, "tag2");

                                        /* Create top level form as a child of root window.
                                        */
form = XtVaCreateWidget("main_window",
xmFormWidgetClass,  toplevel,           /* Parent widget. */
XmNfractionBase,42,
XmNwidth,500,XmNheight, 400,            /* Size of form. */
NULL);
```

```
                                        /* Create frame for machine name, time and date. */
frame_tm = XtVaCreateManagedWidget("frame",
xmFrameWidgetClass, form,               /* Parent widget. */
XmNshadowType, XmSHADOW_ETCHED_OUT,     /* Type of shadow. */
                                        /* Attached positions with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 7,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 11,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 41, NULL);

                                        /* Create form widget for machine name, time and
                                        date.
                                        */
form_tm = XtVaCreateManagedWidget("main_window",
xmFormWidgetClass,  frame_tm,           /* Parent widget. */
XmNfractionBase, 3,                     /* No. Of row and column. */
NULL);

                                        /* Create machine name's label and attach it to the
                                        form widget.
                                        */
XtVaCreateManagedWidget("Machine: okstate", /* Machine name. */
xmLabelWidgetClass, form_tm,            /* Parent widget. */
                                        /* Attached positions with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 3,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 1,
XmNfontList, fontlist,                  /* Use the user defined fonts for the text. */
NULL);

                                        /* Create a drawing area widget for display time and
                                        date.
                                        */
draw2 = XtVaCreateManagedWidget("draw_tm",
xmDrawingAreaWidgetClass, form_tm,      /* Parent widget. */
                                        /* Attached positions with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 3,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 3,
XmNwidth,12, XmNheight, 50, NULL);

                                        /* Get XLib Display Screen and Window ID's  for
                                        drawing area widget.
                                        */
display2 = XtDisplay(draw2);
screen_ptr = XtScreen(draw2);

                                        /* Set the drawing area widget as the "work area" of
                                        main window.
                                        */
XtVaSetValues(form_tm, XmNworkWindow, draw2, NULL);

                                        /* Set callback routine for exposure event for the
                                        drawing area.
                                        */
XtAddCallback(draw2, XmNexposeCallback, draw_cbk, toplevel);

                                        /* Get the foreground pixel's color. */
gcv.foreground = BlackPixelOfScreen(screen_ptr);
                                        /* Create graphics context and pass it to the call
                                        back routine as a user data. The graphics context
                                        also contain foreground color informatin.
                                        */
gc2 = XCreateGC(display2, RootWindowOfScreen(screen_ptr), GCForeground, &gcv);

                                        /* Create frame widget for push buttons on the main
                                        window.
                                        */
frame = XtVaCreateWidget("frame",
xmFrameWidgetClass, form,               /* Parent widget. */
XmNshadowType, XmSHADOW_ETCHED_OUT,
                                        /* Attached positions with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 13,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 41,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 11, NULL);
```

```
                                        /* Create form widget for push buttons. */
form5 = XtVaCreateWidget("main_window",
xmFormWidgetClass,   frame,
XmNfractionBase, 21, NULL);

                                        /* Create CPU Activities push button and attached it
                                        to the form5.
                                        */
wid_cpu=XtVaCreateManagedWidget ("CPU\nACTIVITIES",
xmPushButtonWidgetClass, form5,         /* Parent widget. */
                                        /* Attached positions with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 1,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 5,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 20, NULL);

                                        /* Set call back routine to the push button. Call
                                        back routine is cpuwin_callback.
                                        */
XtAddCallback(wid_cpu, XmNactivateCallback, cpuwin_callback, NULL);

                                        /* Create System Counters push button and attached
                                        it to the form5.
                                        */
Sys_ctr=XtVaCreateManagedWidget ("SYSTEM\nCOUNTERS",
xmPushButtonWidgetClass, form5,         /* Parent widget. */
                                        /* Attached positions with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 6,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 10,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 20, NULL);

                                        /* Set call back routine to the push button. Call
                                        back routine is counter_callback.
                                        */
XtAddCallback(Sys_ctr, XmNactivateCallback, counter_callback, NULL);

                                        /* Create the help button and attached it to the
                                        form5.
                                        */
Help=XtVaCreateManagedWidget ("HELP",
xmPushButtonWidgetClass, form5,         /* Parent widget. */
                                        /* Attached positions with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 11,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 15,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 20, NULL);

                                        /* Set the user data. */
XtVaSetValues(Help, XmNuserData, 0, NULL);

                                        /* Set call back for help push button. */
XtAddCallback(Help, XmNactivateCallback, help_cb, NULL);

                                        /* Create the exit button and attached it to the
                                        form5.
                                        */
exitb=XtVaCreateManagedWidget ("EXIT",
xmPushButtonWidgetClass, form5,         /* Parent widget. */
                                        /* Attached positions with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 16,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 20,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 20, NULL);

                                        /* Set call back for exit push button. */
XtAddCallback(exitb, XmNactivateCallback, my_callback, NULL);

                                        /* Manage form5 and its children. It is necessary to
                                        make it appear on the screen.
                                        */
XtManageChild(form5);

                                        /* Create frame for the load averages. */
frame2 = XtVaCreateWidget("frame2",
xmFrameWidgetClass, form,               /* Parent widget. */
XmNshadowType, XmSHADOW_ETCHED_OUT,     /* Shadow type. */
                                        /* Attached positions with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 13,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 41,
```

```
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 13,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 41, NULL);

                                        /* Create form for the load averages. */
form2 = XtVaCreateWidget("form2",
xmFormWidgetClass,  frame2,             /* Parent widget. */
XmNfractionBase, 30,                    /* No. Of row and column. */
XmNwidth,500, XmNheight, 500, NULL);

                                        /* Create a drawing area widget for load averages.
                                        */
draw = XtVaCreateWidget("draw",
xmDrawingAreaWidgetClass, form2,        /* Parent widget. */
                                        /* Attached positions with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 27,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 2,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 30,
XmNwidth,500, XmNheight, 500, NULL);

                                        /* Get XLib Display Screen and Window ID's  for
                                        draw. */
display = XtDisplay(draw);
screen_ptr = XtScreen(draw);

                                        /* Get color map of the drawing area for using
                                        color. */
cmap=DefaultColormapOfScreen(XtScreen(draw));

                                        /* Assign a color to the drawing area. */
XAllocNamedColor(display,cmap,"black",&xcolour,&spare);
fill_pixel=xcolour.pixel;               /* Save this color. */

                                        /* Set the Drawing Area as the "work area" of main
                                        window.
                                        */
XtVaSetValues(form2, XmNworkWindow, draw, NULL);

                                        /* Add callback for exposure event. */
XtAddCallback(draw, XmNexposeCallback, draw_cbk, toplevel);

                                        /* Create a GC. Attach GC to the Drawing Area's
                                        XmNuserData. This is a useful method to pass data.
                                        */
gcv.foreground = BlackPixelOfScreen(screen_ptr);
gc = XCreateGC(display, RootWindowOfScreen(screen_ptr), GCForeground, &gcv);

                                        /* Set the user data. */
XtVaSetValues(draw, XmNuserData, gc, NULL);

                                        /* Manage the drawing area. */
XtManageChild(draw);

                                        /* Create frame for labels of drawing area. */
frame4 = XtVaCreateManagedWidget("frame4",
xmFrameWidgetClass, form2,
XmNshadowType, XmSHADOW_ETCHED_OUT,
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 27,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 30,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 30, NULL);

                                        /* Create form for labels of drawing area. */
form3 = XtVaCreateManagedWidget("form2",
xmFormWidgetClass,  frame4,
XmNfractionBase, 10, NULL);

                                        /* Create label for load averages. */
last_lb[0]=XtVaCreateManagedWidget("LAST:",
xmLabelWidgetClass, form3,
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 10,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 1, NULL);

                                        /* Create label for load averages. */
last_lb[1]=XtVaCreateManagedWidget("1min.",
xmLabelWidgetClass, form3,
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 10,
```

```
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 2,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 3, NULL);

                                        /* Create label for load averages. */
last_lb[2]=XtVaCreateManagedWidget("5min.",
xmLabelWidgetClass, form3,
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 10,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 5,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 7, NULL);

                                        /* Create label for load averages. */
last_lb[3]=XtVaCreateManagedWidget("15min.",
xmLabelWidgetClass, form3,
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 10,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 8,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 10,
XmNalignment,XmALIGNMENT_BEGINNING,/* Left justification */ NULL);

                                        /* Create a label for y-axis of the load averages.
                                        */
lb_load_name=XtVaCreateManagedWidget(lable_name,
xmLabelWidgetClass, form2,
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 27,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 2,
XmNalignment,XmALIGNMENT_CENTER, NULL);

                                        /* Create frame for title of xmon program. */
frame3 = XtVaCreateWidget("frame3",
xmFrameWidgetClass, form,
XmNshadowType, XmSHADOW_ETCHED_OUT,
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 1,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 6,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 41, NULL);


                                        /* Create label for title of xmon program. */
XtVaCreateManagedWidget("title",
xmLabelWidgetClass, frame3,
XmNlabelString, Title_st,               /* Title string. */
NULL);

                                        /* Manage all remaining widgets. */
XtManageChild(frame1);
XtManageChild(frame3);
XtManageChild(form2);
XtManageChild(frame2);
XtManageChild(form);
                                        /* Add a background process to X Window. This
                                        background process gathers the data of system load
                                        averages, draw the bar graphs of load averages on
                                        the drawing area of the main window, and display
                                        current date and time. This process name is 'pnt'.
                                        */
XtAppAddWorkProc(app2,pnt,draw);

                                        /* Display root window and all its children. */
XtRealizeWidget (toplevel);

                                        /* Enter in the main loop and process the x events.
                                        */
XtAppMainLoop (app2);
                        }
                )
        }                               /* Parent. */
}                                       /* Main routine. */


                                        /* This routine executes as a background process.
                                        The jobs of this routine is to gather the system
                                        load averages, draw the bar graphs of load averages
                                        on the drawing area of the main window, and display
                                        current date and time. */
void pnt(draw)
Widget draw;
{
```

```
int x,y,x_end,y_end,Base_y=240;        /* Coordinates of drawing area. */
int  a,b,c;
float read_value,temp;
char valst[7];
char str1[6];
GC gc;
Window win = XtWindow(draw);           /* Get the window ID of the drawing area for load
                                       averages.
                                       */
Window win2 = XtWindow(draw2);         /* Get the window ID of the drawing area for time &
                                       date.
                                       */
                                       /* Select the interested X events for the main
                                       window. The events are mouse buttons press and
                                       release and graphics exposure. Other events are not
                                       interested and rejected by the main window.
                                       */
if(input_select2==0)                   /* If main window is mapped. */
{
input_select2=1;
                                       /* Select the events. */
XSelectInput(dpy_main,XtWindow(toplevel),ButtonPressMask|ButtonReleaseMask|ExposureMask);
}

XClearWindow(display,win);             /* Clear the load averages area. */
XClearWindow(display2,win2);           /* Clear the time and date area. */
XtVaGetValues(draw, XmNuserData, &gc, NULL);

                                       /* Read the load average for last 1 minute from
                                       pipe. */
read(p_main[0],&read_value,sizeof(read_value));
                                       /* Adjust it to fit on the drawing area. */
temp=read_value;
temp=temp*100.0;
y=Base_y-(int)(temp+5);
a=y;
sprintf(valst,"%2.2f",read_value);

                                       /* Set the foreground color. */
XSetForeground(display,gc,fill_pixel);

                                       /* Draw bar graph for load average during last 1
                                       minute.
                                       */
XFillRectangle(display,win,gc,50,y+4,12,Base_y);
                                       /* Display the values. */
XDrawString(display,win,gc,50,y-8,valst,strlen(valst));

                                       /* Read the load average for last 5 minutes from
                                       pipe.
                                       */
read(p_main[0],&read_value,sizeof(read_value));
                                       /* Adjust it to fit on the drawing area. */
temp=read_value;
temp=temp*100.0;
y=Base_y-(int)(temp+5);
b=y;
sprintf(valst,"%2.2f",read_value);

                                       /* Draw bar graph for load average during last 5
                                       minutes.
                                       */
XFillRectangle(display,win,gc,150,y+4,12,Base_y);
                                       /* Display the values. */
XDrawString(display,win,gc,150,y-8,valst,strlen(valst));
                                       /* Read the load average for last 15 minutes from
                                       pipe.
                                       */
read(p_main[0],&read_value,sizeof(read_value));

                                       /* Adjust it to fit on the drawing area. */
temp=read_value;
temp=temp*100.0;
y=Base_y-(int)(temp+5);
c=y;
sprintf(valst,"%2.2f",read_value);
                                       /* Draw bar graph for load average during last 15
                                       minutes.
                                       */
XFillRectangle(display,win,gc,250,y+4,12,Base_y);
```

```
                                        /* Display the values. */
XDrawString(display,win,gc,250,y-8,valst,strlen(valst));
                                        /* Calculate current date and time. */
tim=time(0);
today=localtime(&tim);
ascftime (time_buf, "%a, %b. %d, %y %r", today);
                                        /* Display date and time. */
XDrawString(display2,win2,gc2,23,21,time_buf,strlen(time_buf));
                                        /* Update the display. */
XmUpdateDisplay(toplevel);
}
                                        /* This routine executes as a child process of the
                                        main program. The jobs of this routine is to gather
                                        the system load averages and send them to the main
                                        program through pipe. */
void child_load()
{
FILE *fp;
int n,l;
float a,b,c,ans;
char line[MAX_ITM];                     /* Maximum items in the line which is read by
                                        executing the uptime command.
                                        */

while(1)
{
                                        /* Execute uptime command and redirect its output to
                                        a pipe.
                                        */
        if( (fp=popen("uptime","r"))==NULL)
        {printf("popen error"); exit(0);}
                                        /* Read the output. */
        fgets(line,MAX_ITM,fp);
        l=0;
        n=strlen(line);
        for(l=0;l<=n;l++)
        {                               /* Extract the load values from the command output.
                                        */
                if( (line[l]=='e')&&(line[l+1]==':') )
                {
                        l=l+3;
                                        /* Convert ASCII output to integer. */
                        a=cvt( (int)line[l]);l=l+2;
                        b=cvt( (int)line[l++]);
                        c=cvt( (int)line[l]);
                        ans=(a+(b/10)+(c/100));
                                        /* Send the value of load average during last 1
                                        minute to main program.
                                        */
                        write(p_main[1],&ans,sizeof(ans));
                        l=l+3;
                                        /* Convert ASCII output to integer. */
                        a=cvt( (int)line[l]);l=l+2;
                        b=cvt( (int)line[l++]);
                        c=cvt( (int)line[l] );
                        ans=(a+(b/10)+(c/100));
                                        /* Send the value of load average during last 5
                                        minutes to main program.
                                        */
                        write(p_main[1],&ans,sizeof(ans));
                        l=l+3;
                                        /* Convert ASCII output to integer. */
                        a=cvt( (int)line[l] );l=l+2;
                        b=cvt( (int)line[l++] );
                        c=cvt( (int)line[l] );
                        ans=(a+(b/10)+(c/100));
                                        /* Send the value of load average during last 15
                                        minutes to main program.
                                        */
                        write(p_main[1],&ans,sizeof(ans));
                }
        }                               /* End for. */
        pclose(fp);

}                                       /*End while. */
}
```

```
                                        /* This routine converts the ASCII values to
                                        corresponding integer values so that these values
                                        can be placed into screen line data structure. */
int cvt(int v)
{
int cn;

                switch(v)
                {
                        case 48:        /* If the ASCII value is 48, return 0. */
                        cn=0;
                        break;

                        case 49:        /* If the ASCII value is 49, return 1. */
                        cn=1;
                        break;

                        case 50:        /* If the ASCII value is 50, return 2. */
                        cn=2;
                        break;

                        case 51:        /* If the ASCII value is 51, return 3. */
                        cn=3;
                        break;

                        case 52:        /* If the ASCII value is 52, return 4. */
                        cn=4;
                        break;

                        case 53:        /* If the ASCII value is 53, return 5. */
                        cn=5;
                        break;

                        case 54:        /* If the ASCII value is 54, return 6. */
                        cn=6;
                        break;

                        case 55:        /* If the ASCII value is 55, return 7. */
                        cn=7;
                        break;

                        case 56:        /* If the ASCII value is 56, return 8. */
                        cn=8;
                        break;

                        case 57:        /* If the ASCII value is 57, return 9. */
                        cn=9;
                        break;

                        default:        /* Default value is -1. */
                        cn=-1;
                        break;

                }                       /* end switch. */
        return(cn);
}
                                        /* This is the call back routine for the EXIT
                                        button. When the user presses the EXIT button then
                                        this routine executes. */
void
my_callback(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        printf("Quiting Program\n");    /* Print the message. */
        kill(-1,SIGINT);                /* Kill all child processes. */
        exit(0);
}

                                        /* This is the call back routine for the CPU
                                        ACTIVITIES button. When the user presses this button
                                        then this routine executes. */


void
cpuwin_callback(w, client_data, call_data)
Widget w;
XtPointer client_data;
```

```
XtPointer call_data;
{
int cpu_start=20;
int off=0;
static int chk_btn1=0;
Cursor cursor;
XSetWindowAttributes attrs;

        if(chk_btn1==0)                 /* If CPU window is already mapped then do nothing.
                                        */
        {
                chk_btn1=1;
                                        /* Change the cursor to busy cursor. */
                cursor=XCreateFontCursor(dpy_main,XC_watch);
                attrs.cursor=cursor;
                XChangeWindowAttributes(dpy_main,XtWindow(toplevel),CWCursor,&attrs);

                                        /* Initiate the CPU window. */
                write(cpu_pipe[1],&cpu_start,sizeof(cpu_start));
                sleep(4);
                                        /* Reset the cursor to normal. */
                attrs.cursor=None;
                XChangeWindowAttributes(dpy_main,XtWindow(toplevel),CWCursor,&attrs);
        }
}


#include "cpu_window"                   /* Routines for cpu window. */
#include "counter_window"               /* Routines for system counter. window */


                                        /* This is the call back routine for the SYSTEM
                                        COUNTERS button. When the user presses this button
                                        then this routine executes. */
void
counter_callback(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
int counter_start=25;
static int chk_btn2=0;
Cursor cursor;
XSetWindowAttributes attrs;

        if(chk_btn2==0)                 /* If system counters window is already mapped then
                                        do nothing.
                                        */
        {
                chk_btn2=1;
                                        /* Change the cursor to busy cursor. */
                 cursor=XCreateFontCursor(dpy_main,XC_watch);
                 attrs.cursor=cursor;
                 XChangeWindowAttributes(dpy_main,XtWindow(toplevel), CWCursor,&attrs);
                                        /* Initiate the system counters window. */
                write(counter_pipe[1],&counter_start,sizeof(counter_start));
                sleep(4);
                                        /* Reset the cursor to normal shape. */
                 attrs.cursor=None;
                 XChangeWindowAttributes(dpy_main,XtWindow(toplevel), CWCursor,&attrs);
        }
}
                                        /* This routine updates the  any drawing area
                                        whenever the expose event occurs. */
void
draw_cbk(Widget w, XtPointer data,
          XmDrawingAreaCallbackStruct *cbk)
{
        GC gc;                          /* Graphics context values. */
        Widget TOPwid=(Widget)data;
        XmUpdateDisplay(TOPwid);        /* Update the display. */
}
  /*****************************************************************/
 /*                          File: button_header                    */
/*****************************************************************/

                                        /* This file contains prototypes of call backs
                                        routines for system counter window. */
                                        /* Prototype for call back routines for system
                                        counter window.  */
```

```
void con_help_disp(Widget w);
void con_help(Widget , XtPointer , XtPointer);
void chg_lab0(Widget , XtPointer , XtPointer);
void chg_lab1(Widget , XtPointer , XtPointer);
void chg_lab2(Widget , XtPointer , XtPointer);
void chg_lab3(Widget , XtPointer , XtPointer);
void chg_lab4(Widget , XtPointer , XtPointer);
void chg_lab5(Widget , XtPointer , XtPointer);
void chg_lab6(Widget , XtPointer , XtPointer);
void chg_lab7(Widget , XtPointer , XtPointer);
void chg_lab8(Widget , XtPointer , XtPointer);
void chg_lab9(Widget , XtPointer , XtPointer);
void chg_lab10(Widget , XtPointer , XtPointer);
void chg_lab11(Widget , XtPointer , XtPointer);
void chg_lab12(Widget , XtPointer , XtPointer);
void chg_lab13(Widget , XtPointer , XtPointer);
void chg_lab14(Widget , XtPointer , XtPointer);
void chg_lab15(Widget , XtPointer , XtPointer);
void chg_lab16(Widget , XtPointer , XtPointer);
void chg_lab17(Widget , XtPointer , XtPointer);
void chg_lab18(Widget , XtPointer , XtPointer);
void chg_lab19(Widget , XtPointer , XtPointer);
void chg_lab20(Widget , XtPointer , XtPointer);
void chg_lab21(Widget , XtPointer , XtPointer);
void chg_lab22(Widget , XtPointer , XtPointer);
void chg_lab23(Widget , XtPointer , XtPointer);
void chg_lab24(Widget , XtPointer , XtPointer);
void chg_lab25(Widget , XtPointer , XtPointer);
void chg_lab26(Widget , XtPointer , XtPointer);
void chg_lab27(Widget , XtPointer , XtPointer);
void chg_lab28(Widget , XtPointer , XtPointer);
void chg_lab29(Widget , XtPointer , XtPointer);
void chg_lab30(Widget , XtPointer , XtPointer);
void chg_lab31(Widget , XtPointer , XtPointer);
void chg_lab32(Widget , XtPointer , XtPointer);
void chg_lab33(Widget , XtPointer , XtPointer);
void chg_lab34(Widget , XtPointer , XtPointer);
void chg_lab35(Widget , XtPointer , XtPointer);
void chg_lab36(Widget , XtPointer , XtPointer);
void chg_lab37(Widget , XtPointer , XtPointer);
void chg_lab38(Widget , XtPointer , XtPointer);
void chg_lab39(Widget , XtPointer , XtPointer);
void chg_lab40(Widget , XtPointer , XtPointer);
void chg_lab41(Widget , XtPointer , XtPointer);
void chg_lab42(Widget , XtPointer , XtPointer);
void chg_lab43(Widget , XtPointer , XtPointer);
void chg_lab44(Widget , XtPointer , XtPointer);
void chg_lab45(Widget , XtPointer , XtPointer);
void chg_lab46(Widget , XtPointer , XtPointer);
void chg_lab47(Widget , XtPointer , XtPointer);
void chg_lab48(Widget , XtPointer , XtPointer);
  /*****************************************************************/
 /*                          File: but_cn_callbacks                 */
/*****************************************************************/
                                    /* This file contains call back routines for push
                                    buttons used in system counters window. */
                                    /* This routine changes cursor shape whenever user
                                    clicks on context sensitive button. */
void
con_help(w, client_data, call_data)
Widget w;
XtPointer client_data;              /* Widget specific data. */
XtPointer call_data;                /* User data. */
{
Cursor cursor;                      /* Cursor structure. */
XSetWindowAttributes attrs;         /* Window structure to change its look. */
Display *dpy=XtDisplay(toplevel_crr);/* Get the display ID of main window. */

        Hflag=1;                    /* Flag to indicate that context sensitive button
                                    has been pressed. */

                                    /* Get the cursor ID for hand shape cursor. */
cursor=XCreateFontCursor(dpy,XC_hand2);
        attrs.cursor=cursor;
                                    /* Set the new cursor to the toplevel window. */
XChangeWindowAttributes(dpy,XtWindow(toplevel_crr),CWCursor,&attrs);
}

                                    /* This routine display the context sensitive help
                                    text as well as change the cursor to wait cursor. */
```

```
void con_help_disp(Widget w)
{
Cursor cursor;
XSetWindowAttributes attrs;
Display *dpy=XtDisplay(toplevel_ctr);

                                        /* Get the cursor ID for watch shape cursor. */
        cursor=XCreateFontCursor(dpy,XC_watch);
        attrs.cursor=cursor;
        XChangeWindowAttributes(dpy,XtWindow(toplevel_ctr),CWCursor,&attrs);

                help_cb2(w);            /* create the help window and display the help text.
                                        */
                Hflag=0;                /* Flag to indicate that context sensitive help
                                        window is mapped on the screen. */

                                        /* Reset the cursor to its normal shape. */
        attrs.cursor=None;
        XChangeWindowAttributes(dpy,XtWindow(toplevel_ctr),CWCursor,&attrs);

}
                                        /* This routine execute whenever user presses
                                        "context switch" button on the system counter
                                        window. */
void
chg_lab0(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
                                        /* If user want context sensitive help then display
                                        it. */
        if(Hflag)
        {
                con_help_disp(w);       /* Display the context sensitive help. */
                return;
        }
                                        /* If the requested system counter is already
                                        displayed then return. */
        if(     (Counter[0].Line==12)&&(Counter[0].Val_pos==1) ||
                (Counter[1].Line==12)&&(Counter[1].Val_pos==1) ||
                (Counter[2].Line==12)&&(Counter[2].Val_pos==1) )
                        return;
                                        /* Move the previous counters towards right side of
                                        the window. */
        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

                                        /* Display the requested system counter at the left
                                        side of window. */
        strcpy(Counter[0].Label,"Context SW");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=12;
        Counter[0].Val_pos=1;

}
                                        /* This routine execute whenever user presses
                                        "Deficit" button on the system counter window. */
void
chg_lab1(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
                                        /* If user want context sensitive help then display
                                        it. */
        if(Hflag)
        {
                con_help_disp(w);       /* Display the context sensitive help. */
                return;
```

```
        }
                                        /* If the requested system counter is already
                                        displayed then return. */
        if(     (Counter[0].Line==9)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==9)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==9)&&(Counter[2].Val_pos==2) )
                        return;

                                        /* Move the previous counters towards right side of
                                        the window. */
        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;
                                        /* Display the requested system counter at the left
                                        side of window. */
        strcpy(Counter[0].Label,"Deficit");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=9;
        Counter[0].Val_pos=2;
}
                                        /* This routine execute whenever user presses "Dirty
                                        Memory" button on the system counter window. */
void
chg_lab2(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
                                        /* If user want context sensitive help then display
                                        it. */
        if(Hflag)
        {
                con_help_disp(w);       /* Display the context sensitive help. */
                return;
        }
                                        /* If the requested system counter is already
                                        displayed then return. */
        if(     (Counter[0].Line==11)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==11)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==11)&&(Counter[2].Val_pos==2) )
                        return;
                                        /* Move the previous counters towards right side of
                                        the window. */
        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

                                        /* Display the requested system counter at the left
                                        side of window. */
        strcpy(Counter[0].Label,"Dirty Memory");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=11;
        Counter[0].Val_pos=2;
}
                                        /* This routine execute whenever user presses "Dirty
                                        Page Reclaims" button on the system counter window.
                                        */
void
chg_lab3(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
```

```
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==0)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==0)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==0)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Dirty Page Recs");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=0;
        Counter[0].Val_pos=2;

}

                                        /* This routine execute whenever user presses "Disk
                                        KB" button on the system counter window. */
void
chg_lab4(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==19)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==19)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==19)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Disk KB");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=19;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses "Disk
                                        Transfer" button on the system counter window. */
void
chg_lab5(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
```

```
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==18)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==18)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==18)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Disk Transfer");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=18;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses
                                        "Execs" button on the system counter window. */
void
chg_lab6(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{

        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==17)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==17)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==17)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Execs");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=17;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses "Fast
                                        Wait" button on the system counter window. */
void
chg_lab7(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
```

```
        }

        if(     (Counter[0].Line==6)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==6)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==6)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Fast Wait");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=6;
        Counter[0].Val_pos=1;
}
                                        /* This routine execute whenever user presses
                                        "Forks" button on the system counter window. */
void
chg_lab8(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==15)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==15)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==15)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Forks");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=15;
        Counter[0].Val_pos=1;

}
                                        /* This routine execute whenever user presses "Free
                                        Memory" button on the system counter window. */
void
chg_lab9(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==10)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==10)&&(Counter[1].Val_pos==2)||
```

```
                (Counter[2].Line==10)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Free Memory");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=10;
        Counter[0].Val_pos=2;

}
                                        /* This routine execute whenever user presses "Free
                                        Page Recs" button on the system counter window. */
void
chg_lab10(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==23)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==23)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==23)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Free Page Recs");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=23;
        Counter[0].Val_pos=1;

}
                                        /* This routine execute whenever user presses "FS
                                        Blk Reads" button on the system counter window. */
void
chg_lab11(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==13)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==13)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==13)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
```

```
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"FS BLK Reads");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=13;
        Counter[0].Val_pos=2;

}
                                        /* This routine execute whenever user presses "FS
                                        BLk Writes" button on the system counter window. */
void
chg_lab12(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }


        if(     (Counter[0].Line==14)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==14)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==14)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"FS BLK Writes");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=14;
        Counter[0].Val_pos=2;

}
                                        /* This routine execute whenever user presses "FS
                                        Read Hit" button on the system counter window. */
void
chg_lab13(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==15)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==15)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==15)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
```

```
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"FS Read Hit");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=15;
        Counter[0].Val_pos=2;

}
                                        /* This routine execute whenever user presses "FS
                                        Write Hit" button on the system counter window. */
void
chg_lab14(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }


        if(     (Counter[0].Line==16)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==16)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==16)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"FS Write Hit");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=16;
        Counter[0].Val_pos=2;
}
                                        /* This routine execute whenever user presses
                                        "Interrupts" button on the system counter window. */
void
chg_lab15(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==13)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==13)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==13)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
```

```
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[2].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Interrupts");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=13;
        Counter[0].Val_pos=1;
}
                                        /* This routine execute whenever user presses "Locks
                                        Used" button on the system counter window. */
void
chg_lab16(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==21)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==21)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==21)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Locks Used");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=21;
        Counter[0].Val_pos=2;
}
                                        /* This routine execute whenever user presses
                                        "Percent Locks Used" button on the system counter
                                        window. */
void
chg_lab17(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==22)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==22)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==22)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Percent Locks Used");
```

```
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=22;
        Counter[0].Val_pos=2;

}
                                        /* This routine execute whenever user presses
                                        "Message Ops" button on the system counter window.
                                        */
void
chg_lab18(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==23)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==23)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==23)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Message Ops");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=23;
        Counter[0].Val_pos=2;

}
                                        /* This routine execute whenever user presses "No.
                                        of Processes" button on the system counter window.
                                        */
void
chg_lab19(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==3)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==3)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==3)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"No. of Processes");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
```

```
        Counter[0].Line=3;
        Counter[0].Val_pos=1;

}
                                        /* This routine execute whenever user presses "Page
                                        Faults" button on the system counter window. */
void
chg_lab20(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==22)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==22)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==22)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Page Faults");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=22;
        Counter[0].Val_pos=1;

}
                                        /* This routine execute whenever user presses "Page
                                        Ins" button on the system counter window. */
void
chg_lab21(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==1)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==1)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==1)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Page Ins");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=1;
        Counter[0].Val_pos=2;
```

```
}

                                        /* This routine execute whenever user presses "Page
                                        Outs" button on the system counter window. */
void
chg_lab22(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==3)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==3)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==3)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Page Outs");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=3;
        Counter[0].Val_pos=2;
}

                                        /* This routine execute whenever user presses "Pages
                                        Paged" In button on the system counter window. */
void
chg_lab23(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==2)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==2)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==2)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Pages Paged In");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=2;
        Counter[0].Val_pos=3;
}
```

```
                                        /* This routine execute whenever user presses 'Pages
                                        Paged Out" button on the system counter window. */
void
chg_lab24(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==4)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==4)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==4)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Pages Paged Out");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=4;
        Counter[0].Val_pos=2;
}
                                        /* This routine execute whenever user presses "Pages
                                        Swapped In" button on the system counter window. */
void
chg_lab25(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==6)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==6)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==6)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Pages Swapped In");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=6;
        Counter[0].Val_pos=2;
}
                                        /* This routine execute whenever user presses "Pages
                                        Swapped Out" button on the system counter window. */
void
chg_lab26(w, client_data, call_data)
Widget w;
XtPointer client_data;
```

```
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==8)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==8)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==8)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Pages Swapped Out");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=8;
        Counter[0].Val_pos=2;
}

                                        /* This routine execute whenever user presses "Procs
                                        FS I/O Wt" button on the system counter window. */
void
chg_lab27(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==9)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==9)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==9)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Procs FS I/O Wt");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=9;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses "Procs
                                        Phy I/O Wt" button on the system counter window. */
void
chg_lab28(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
```

```
                return;
        }

        if(     (Counter[0].Line==10)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==10)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==10)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Procs Phy I/O Wt");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=10;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses "Raw
                                        Reads" button on the system counter window. */
void
chg_lab29(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==17)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==17)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==17)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Raw Reads");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=17;
        Counter[0].Val_pos=2;
}

                                        /* This routine execute whenever user presses "Raw
                                        Writes" button on the system counter window. */
void
chg_lab30(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==18)&&(Counter[0].Val_pos==2)||
```

```
                (Counter[1].Line==18)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==18)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Raw Writes");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=18;
        Counter[0].Val_pos=2;
}

                                        /* This routine execute whenever user presses "Raw
                                        Read KB" button on the system counter window. */
void
chg_lab31(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==19)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==19)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==19)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Raw Read KB");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=19;
        Counter[0].Val_pos=2;
}

                                        /* This routine execute whenever user presses "Raw
                                        Write KB" button on the system counter window. */
void
chg_lab32(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==20)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==20)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==20)&&(Counter[2].Val_pos==2) )
                        return;
```

```
        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Raw Write KB");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=20;
        Counter[0].Val_pos=2;
}

                                        /* This routine execute whenever user presses
                                        "Runnable Procs" button on the system counter
                                        window. */
void
chg_lab33(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==5)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==5)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==5)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Runnable Procs");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=5;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses
                                        "Running Procs" button on the system counter window.
                                        */
void
chg_lab34(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==4)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==4)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==4)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
```

```
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Running Procs");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=4;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses
                                        "Semaphore Ops" button on the system counter window.
                                        */
void
chg_lab35(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==0)&&(Counter[0].Val_pos==3)||
                (Counter[1].Line==0)&&(Counter[1].Val_pos==3)||
                (Counter[2].Line==0)&&(Counter[2].Val_pos==3) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Semaphore Ops");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=0;
        Counter[0].Val_pos=3;
}

                                        /* This routine execute whenever user presses
                                        "Sleeping Procs" button on the system counter
                                        window. */
void
chg_lab36(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==7)&&(Counter[0].Val_pos==1) ||
                (Counter[1].Line==7)&&(Counter[1].Val_pos==1) ||
                (Counter[2].Line==7)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
```

```
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Sleeping Procs");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=7;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses "Swap
                                        Ins" button on the system counter window. */
void
chg_lab37(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==5)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==5)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==5)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Swap Ins");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=5;
        Counter[0].Val_pos=2;
}

                                        /* This routine execute whenever user presses "Swap
                                        Outs" button on the system counter window. */
void
chg_lab38(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==7)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==7)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==7)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
```

```
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Swap Outs");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=7;
        Counter[0].Val_pos=2;
}

                                        /* This routine execute whenever user presses
                                        "Swapped Procs" button on the system counter window.
                                        */
void
chg_lab39(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==8)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==8)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==8)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Swapped Procs");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=8;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses
                                        "System Calls" button on the system counter window.
                                        */
void
chg_lab40(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==11)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==11)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==11)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
```

```
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"System Calls");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=11;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses "Total
                                        Sys Time" button on the system counter window. */
void
chg_lab41(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==1)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==1)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==1)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Total Sys Time");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=1;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses "Total
                                        User Time" button on the system counter window. */
void
chg_lab42(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==0)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==0)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==0)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Total Usr Time");
```

```
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=0;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses "Total
                                        Time" button on the system counter window. */
void
chg_lab43(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==2)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==2)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==2)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Total Time");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=2;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses "Total
                                        Virtual Mem" button on the system counter window. */
void
chg_lab44(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==12)&&(Counter[0].Val_pos==2)||
                (Counter[1].Line==12)&&(Counter[1].Val_pos==2)||
                (Counter[2].Line==12)&&(Counter[2].Val_pos==2) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Total Virtual Mem");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=12;
        Counter[0].Val_pos=2;
```

```
}

                                        /* This routine execute whenever user presses
                                        "Traps" button on the system counter window. */
void
chg_lab45(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==14)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==14)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==14)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"Traps");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=14;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses "TTY
                                        Char In" button on the system counter window. */
void
chg_lab46(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==20)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==20)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==20)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"TTY Char In");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=20;
        Counter[0].Val_pos=1;
```

```
                                        /* This routine execute whenever user presses "TTY
                                        Char Out" button on the system counter window. */
void
chg_lab47(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==21)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==21)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==21)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"TTY Char Out");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=21;
        Counter[0].Val_pos=1;
}

                                        /* This routine execute whenever user presses
                                        "VForks" button on the system counter window. */
void
chg_lab48(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{

        if(Hflag)
        {
                con_help_disp(w);
                return;
        }

        if(     (Counter[0].Line==16)&&(Counter[0].Val_pos==1)||
                (Counter[1].Line==16)&&(Counter[1].Val_pos==1)||
                (Counter[2].Line==16)&&(Counter[2].Val_pos==1) )
                        return;

        strcpy(Counter[2].Label,Counter[1].Label);
        XtVaSetValues(disp_bt3,XmNlabelString,
                        XmStringCreateSimple(Counter[2].Label),NULL);
        Counter[2].Line=Counter[1].Line;
        Counter[2].Val_pos=Counter[1].Val_pos;

        strcpy(Counter[1].Label,Counter[0].Label);
        XtVaSetValues(disp_bt2,XmNlabelString,
                        XmStringCreateSimple(Counter[1].Label),NULL);
        Counter[1].Line=Counter[0].Line;
        Counter[1].Val_pos=Counter[0].Val_pos;

        strcpy(Counter[0].Label,"VForks");
        XtVaSetValues(disp_bt1,XmNlabelString,
                        XmStringCreateSimple(Counter[0].Label),NULL);
        Counter[0].Line=16;
        Counter[0].Val_pos=1;
}
```

```
/*****************************************************************/
/*                          File: store_counter                  */
/*****************************************************************/

                                    /* This routine is used to fill the array with
                                    system counter values to present on the graph.
                                    Variable line is used for screen line number.
                                    Variables V1st and V2nd represent first and 2nd
                                    values on the curses screen. This routine sample 20
                                    values. */

void store_it(int line,int V1st,int V2nd)
{
int k;
                                    /* Determine which screen line's values is going to
                                    store. */
        switch(line)
        {
                case 0:             /* For line 0. */
                vc[0].VFirst[idn0]=V1st; /*Store 1st value in structure.*/
                vc[0].V2nd[idn0++]=V2nd; /*Store 2nd value in structure.*/
                if(idn0 > 20)       /* store 20 values. */
                {
                        for(k=0;k<=19;k++)
                        {           /* Adjust the array of structures. */
                                vc[0].VFirst[k]=vc[0].VFirst[k+1];
                                vc[0].V2nd[k]=vc[0].V2nd[k+1];
                        }
                        idn0=20;    /* Set array index to value 20. */
                }
                break;

                case 1:             /* For line 1. */
                vc[1].VFirst[idn1]=V1st;
                vc[1].V2nd[idn1++]=V2nd;
                if(idn1 > 20)
                {
                        for(k=0;k<=19;k++)
                        {
                                vc[1].VFirst[k]=vc[1].VFirst[k+1];
                                vc[1].V2nd[k]=vc[1].V2nd[k+1];
                        }
                        idn1=20;
                }
                break;

                case 2:             /* For line 2. */
                vc[2].VFirst[idn2]=V1st;
                vc[2].V2nd[idn2++]=V2nd;
                if(idn2 > 20)
                {
                        for(k=0;k<=19;k++)
                        {
                                vc[2].VFirst[k]=vc[2].VFirst[k+1];
                                vc[2].V2nd[k]=vc[2].V2nd[k+1];
                        }
                        idn2=20;
                }
                break;

                case 3:             /* For line 3. */
                vc[3].VFirst[idn3]=V1st;
                vc[3].V2nd[idn3++]=V2nd;
                if(idn3 > 20)
                {
                        for(k=0;k<=19;k++)
                        {
                                vc[3].VFirst[k]=vc[3].VFirst[k+1];
                                vc[3].V2nd[k]=vc[3].V2nd[k+1];
                        }
                        idn3=20;
                }
                break;

                case 4:             /* For line 4. */
                vc[4].VFirst[idn4]=V1st;
                vc[4].V2nd[idn4++]=V2nd;
                if(idn4 > 20)
                {
```

```
        for(k=0;k<=19;k++)
        {
                vc[4].VFirst[k]=vc[4].VFirst[k+1];
                vc[4].V2nd[k]=vc[4].V2nd[k+1];
        }
        idn4-20;
}
break;

case 5:                 /* For line 5. */
vc[5].VFirst[idn5]=V1st;
vc[5].V2nd[idn5++]=V2nd;
if(idn5 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[5].VFirst[k]=vc[5].VFirst[k+1];
                vc[5].V2nd[k]=vc[5].V2nd[k+1];
        }
        idn5=20;
}
break;

case 6:                 /* For line 6. */
vc[6].VFirst[idn6]=V1st;
vc[6].V2nd[idn6++]=V2nd;
if(idn6 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[6].VFirst[k]=vc[6].VFirst[k+1];
                vc[6].V2nd[k]=vc[6].V2nd[k+1];
        }
        idn6=20;
}
break;

case 7:                 /* For line 7. */
vc[7].VFirst[idn7]=V1st;
vc[7].V2nd[idn7++]=V2nd;
if(idn7 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[7].VFirst[k]=vc[7].VFirst[k+1];
                vc[7].V2nd[k]=vc[7].V2nd[k+1];
        }
        idn7=20;
}
break;

case 8:                 /* For line 8. */
vc[8].VFirst[idn8]=V1st;
vc[8].V2nd[idn8++]=V2nd;
if(idn8 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[8].VFirst[k]=vc[8].VFirst[k+1];
                vc[8].V2nd[k]=vc[8].V2nd[k+1];
        }
        idn8=20;
}
break;

case 9:                 /* For line 9. */
vc[9].VFirst[idn9]=V1st;
vc[9].V2nd[idn9++]=V2nd;
if(idn9 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[9].VFirst[k]=vc[9].VFirst[k+1];
                vc[9].V2nd[k]=vc[9].V2nd[k+1];
        }
        idn9=20;
}
break;

case 10:                /* For line 10. */
```

```
vc[10].VFirst[idn1_].Vlst;
vc[10].V2nd[idn10++]=V2nd;
if(idn10 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[10].VFirst[k]=vc[1^].VFirst[k+1];
                vc[10].V2nd[k]=vc[10].V2nd[k+1];
        }
        idn10=2( ;
}
break;

case 11:                /* For line 11. */
vc[11].VFirst[idn11]=Vlst;
vc[11].V2nd[idn11++]=V2nd;
if(idn11 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[11].VFirst[k]=vc[11].VFirst[k+1];
                vc[11].V2nd[k]=vc[11].V2nd[k+1];
        }
        idn11=20;
}
break;

case 12:                /* For line 12. */
vc[12].VFirst[idn12]=Vlst;
vc[12].V2nd[idn12++]=V2nd;
if(idn12 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[12].VFirst[k]=vc[12].VFirst[k+1];
                vc[12].V2nd[k]=vc[12].V2nd[k+1];
        }
        idn12=20;
}
break;

case 13:                /* For line 13. */
vc[13].VFirst[idn13]=Vlst;
vc[13].V2nd[idn13++]=V2nd;

if(idn13 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[13].VFirst[k]=vc[13].VFirst[k+1];
                vc[13].V2nd[k]=vc[13].V2nd[k+1];
        }
        idn13=20;

break;

case 14:                /* For line 14. */
vc[14].VFirst[idn14]=Vlst;
vc[14].V2nd[idn14++]=V2nd;
if(idn14 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[14].VFirst[k]=vc[14].VFirst[k+1];
                vc[14].V2nd[k]=vc[14].V2nd[k+1];
        }
        idn14=20;
}
break;

case 15:                /* For line 15. */
vc[15].VFirst[idn15]=Vlst;
vc[15].V2nd[idn15++]=V2nd;
if(idn15 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[15].VFirst[k]=vc[15].VFirst[k+1];
                vc[15].V2nd[k]=vc[15].V2nd[k+1];
        }
```

```
        idn15=20;
}
break;

case 16:                /* For line 16. */
vc[16].VFirst[idn16]=V1st;
vc[16].V2nd[idn16++]=V2nd;
if(idn16 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[16].VFirst[k]=vc[16].VFirst[k+1];
                vc[16].V2nd[k]=vc[16].V2nd[k+1];
        }
        idn16=20;
}
break;

case 17:                /* For line 17. */
vc[17].VFirst[idn17]=V1st;
vc[17].V2nd[idn17++]=V2nd;
if(idn17 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[17].VFirst[k]=vc[17].VFirst[k+1];
                vc[17].V2nd[k]=vc[17].V2nd[k+1];
        }
        idn17=20;
}
break;

case 18:                /* For line 18. */
vc[18].VFirst[idn18]=V1st;
vc[18].V2nd[idn18++]=V2nd;
if(idn18 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[18].VFirst[k]=vc[18].VFirst[k+1];
                vc[18].V2nd[k]=vc[18].V2nd[k+1];
        }
        idn18=20;
}
break;

case 19.                /* For line 19. */
vc[19].VFirst[idn19]=V1st;
vc[19].V2nd[idn19++]=V2nd;
if(idn19 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[19].VFirst[k]=vc[19].VFirst[k+1];
                vc[19].V2nd[k]=vc[19].V2nd[k+1];
        }
        idn19=20;
}
break;

case 20:                /* For line 20. */
vc[20].VFirst[idn20]=V1st;
vc[20].V2nd[idn20++]=V2nd;
if(idn20 > 20)
{
        for(k=0;k<=19;k++)
        {
                vc[20].VFirst[k]=vc[20].VFirst[k+1];
                vc[20].V2nd[k]=vc[20].V2nd[k+1];
        }
        idn20=20;
}
break;

case 21:                /* For line 21. */
vc[21].VFirst[idn21]=V1st;
vc[21].V2nd[idn21++]=V2nd;
if(idn21 > 20)
{
        for(k=0;k<=19;k++)
```

```
                        {
                                vc[21].VFirst[k]=vc[21].VFirst[k+1];
                                vc[21].V2nd[k]=vc[21].V2nd[k+1];
                        }
                        idn21=20;
                }
                break;

                case 22:                /* For line 22. */
                vc[22].VFirst[idn22]=V1st;
                vc[22].V2nd[idn22++]=V2nd;
                if(idn22 > 20)
                {
                        for(k=0;k<=19;k++)
                        {
                                vc[22].VFirst[k]=vc[22].VFirst[k+1];
                                vc[22].V2nd[k]=vc[22].V2nd[k+1];
                        }
                        idn22=20;
                }
                break;

                case 23:                /* For line 23. */
                vc[23].VFirst[idn23]=V1st;
                vc[23].V2nd[idn23++]=V2nd;
                if(idn23 > 20)
                {
                        for(k=0;k<=19;k++)
                        {
                                vc[23].VFirst[k]=vc[23].VFirst[k+1];
                                vc[23].V2nd[k]=vc[23].V2nd[k+1];
                        }
                        idn23=20;
                }
                break;
        }                               /* Switch end. */
}


  /*****************************************************************/
 /*                          File: counter_window                 */
/*****************************************************************/


                                        /* This routine executes as a child process of the
                                        main program. The purpose of this routine is to
                                        construct a system counter window, map it on the
                                        screen, fork a child process to capture the system
                                        counter values, take user input, and display system
                                        counter graphs. */
void child3(argc,argv)
int argc;
char *argv[];
{
                                        /* Declearation of widgets for system counter
                                        window. */
    Widget  form,frame_tm,frame_lb,form5,form6,draw;
    Widget help,rowcol,bttn[60];
    Widget frame_sc1,frame_sc2,frame_sc3,fr1,fr2,fr3,frame_top,form_top;
    Widget Help_bttn;

                                        /* Labels for system counter buttons. */
        char *label_name'}={"Context SW","Deficit","Dirty Memory",
                        "Dirty Page Recs","Disk KB","Disk Transf","Execs",
                        "Fast Wait","Forks","Free Memory","Free Page Recs",
                        "FS Blk Reads","FS Blk Writes","FS Read Hit",
                        "FS Write Hit","Interrupts","Locks Used",
                        "% Locks Used","Message Ops","No. of Procs",
                        "Page Faults","Page Ins","Page Outs","Pages Paged In",
                        "Pages Paged Out","Pages Swapped In",
                        "Pages Swapped Out","Procs FS io Wt","Pro Phy io Wt",
                        "Raw Reads","Raw Writes","Raw Read KB","Raw Write KB",
                        "Runnable Procs","Running Procs","Semaphore Ops",
                        "Sleeping Procs","Swap Ins","Swap Outs",
                        "Swapped Procs","System Calls","Total Sys Time",
                        "Total User Time","Total Time","Total Virtual Mem",
                        "Traps","TTY Char In","TTY Char Out","VForks"};
```

```
                                /* Graphics context for drawing area used to diplay
                                the system counter graphs. */
        XGCValues gcv;
        GC gc;
                                /* Motif string type for button label. */
        XmString label_string,button_string[60];
        XmString label_string_Max1,label_string_Max2,label_string_Max3,
                 label_string_Min1,label_string_Min2,label_string_Min3;


        XtAppContext app;       /* Application context for system counter window. */
        XEvent event;           /* User gererated events. */
        int kk,i,ad=2;
        int pid1,pid2;          /* Process ID for child process. */
        int GetData=0;          /* Flag to start child process. */
        int counter_start=0;
        Cursor cursor;          /* Cursor structure to change cursor shape. */
        XSetWindowAttributes attrs;   /* Window attribute's structure. */

        Font font;              /* X Window's font structure to set different font.
                                */
        XmFontList fontlist;    /* Add different fonts to the font's list. */
        initialize_graph();     /* Initialize graph structures and counters
                                structures. */

                                /* create Motif string data type by simple character
                                string to use as a widget labels. Toral 49 system
                                counters buttons. */
        for(kk=0;kk<=48;kk++)
                button_string[kk]=XmStringCreateSimple(label_name[kk]);

        pid1=fork();            /* Create a child process. */
        if(pid1==0)
        {
                child1();       /* Gather system counters data and send them through
                                pipe. */
        }
        else
        {                       /* Parent process. */

                                /* Create root window by initialization of X window.
                                */
                toplevel_ctr = XtVaAppInitialize (&app, "Demos",
                               NULL, 0, &argc, argv,
                               NULL, NULL);
                                /* Get display ID of root window. */
                dpy_ctr=XtDisplay(toplevel_ctr);

                                /* Create form widget as a child of root window. All
                                other widgets are the children of this form. */
form = XtVaCreateWidget("main_window", xmFormWidgetClass,
toplevel_ctr,                   /* Parent. */
XmNfractionBase, 42,            /* No. of row and column of the form. */
XmNwidth,830, XmNheight, 62 ,   /* Size of form. */
NULL);

                                /* Create a frame widget on the form widget to
                                contain the system counters buttons widgets. */
frame_top = XtVaCreateManagedWidget("frame_top", xmFrameWidgetClass,
form,                           /* Parent widget. */
XmNshadowType, XmSHADOW_ETCHED_OUT,   /* Type of shadow to give 3D look. */
XmNtopAttachment,               /* Attatche positions of frame with the form. */
XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION,XmNbottomPosition, 16,
XmNleftAttachment, XmATTACH_POSITION,XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION,XmNrightPosition, 42,NULL);


                                /* Create form for system counters buttons. */
form_top = XtVaCreateManagedWidget("form_top",
xmFormWidgetClass,   frame_top,   /* Parent. */
XmNfractionBase, 20,            /* No. of row and column of the form. */
XmNwidth,810,XmNheight, 100,NULL);

                                /* Create row and column widget for the base of all
                                the system counters buttons. */
rowcol = XtVaCreateManagedWidget("rowcol",
xmRowColumnWidgetClass, form_top,     /* Parent. */
XmNpacking,     XmPACK_COLUMN,        /* Layout style of all buttons on row column widget.
                                */
```

```
XmNnumColumns,8,                        /* No. of columns. */
XmNorientation, XmVERTICAL,             /* Buttons orientation. */
                                        /* Attatche positions of row and column widget with
                                        the form. */
XmNtopAttachment, XmATTACH_POSITION,XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION,XmNbottomPosition, 17,
XmNleftAttachment, XmATTACH_POSITION,XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION,XmNrightPosition, 20,NULL);

                                        /* Create a information label. */
XtVaCreateManagedWidget("Click above buttons to display\nsystem counters.",
xmLabelWidgetClass,form_top,            /* Parent. */
                                        /* Attatche positions of label widget with the form.
                                        */
XmNtopAttachment, XmATTACH_POSITION,XmNtopPosition, 17,
XmNbottomAttachment, XmATTACH_POSITION,XmNbottomPosition, 20,
XmNleftAttachment, XmATTACH_POSITION,XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION,XmNrightPosition, 8,
XmNalignment,XmALIGNMENT_BEGINNING,     /* Text on the label is left justified. */
NULL);

                                        /* Create a information label. */
XtVaCreateManagedWidget("Press Alt+Tab to go back\nto Main window.",
xmLabelWidgetClass,form_top,            /* Parent. */
                                        /* Attatche positions of label widget with the form.
                                        */
XmNtopAttachment, XmATTACH_POSITION,XmNtopPosition, 17,
XmNbottomAttachment, XmATTACH_POSITION,XmNbottomPosition, 20,
XmNleftAttachment, XmATTACH_POSITION,XmNleftPosition, 15,
XmNrightAttachment, XmATTACH_POSITION,XmNrightPosition, 20,
XmNalignment,XmALIGNMENT_BEGINNING,     /* Text on the label is left justified. */
NULL);


                                        /* Create a context sensitive help button. */
Help_bttn=XtVaCreateManagedWidget ("Context Sensitive Help",
xmPushButtonWidgetClass, form_top,      /* Parent. */
/* Attatche positions of button widget with the form. */
XmNtopAttachment, XmATTACH_POSITION,XmNtopPosition, 17,
XmNbottomAttachment, XmATTACH_POSITION,XmNbottomPosition, 20,
XmNleftAttachment, XmATTACH_POSITION,XmNleftPosition, 8,
XmNrightAttachment, XmATTACH_POSITION,XmNrightPosition, 12,NULL);

                                        /* Set the background color of context sensitive
                                        help button to yellow. */
XtVaSetValues(Help_bttn,XtVaTypedArg,XmNbackground,XmRString,"Yellow",7,NULL);

                                        /* Set call back routine for context sensitive help
                                        button. Call back routine is Help_bttn. */
XtAddCallback(Help_bttn,XmNactivateCallback,con_help,NULL);

                                        /* Create push buttons for system counters. */
for(kk=0;kk<=48;kk++){
bttn[kk]=XtVaCreateManagedWidget ("button",xmPushButtonWidgetClass, rowcol,
XmNlabelString, button_string[kk],      /* Labels for push buttons. */
XmNalignment,XmALIGNMENT_CENTER,        /* Text on the label is center justified. */
NULL);
                                        /* Set the background color of system counter
                                        buttons to green. */
XtVaSetValues(bttn[kk],XtVaTypedArg,XmNbackground,XmRString,"green",6,NULL);
}
                                        /* Set user data for each push button. */
for(kk=0;kk<=48;kk++)
XtVaSetValues(bttn[kk], XmNuserData, kk, NULL);
                                        /* Destroy Motif string data because we don't need
                                        it any more. It consume lot of memory. */
for(kk=0;kk<=48;kk++)
XmStringFree(button_string[kk]);
                                        /* Set call back routines for each push button.
                                        These call back routines execute whenever the user
                                        presses any system counter button. The number for
                                        button of context switches is 0 and for Vforks is
                                        48. */
XtAddCallback(bttn[0],XmNactivateCallback,chg_lab0,NULL);
XtAddCallback(bttn[1],XmNactivateCallback,chg_lab1,NULL);
XtAddCallback(bttn[2],XmNactivateCallback,chg_lab2,NULL);
XtAddCallback(bttn[3],XmNactivateCallback,chg_lab3,NULL);
XtAddCallback(bttn[4],XmNactivateCallback,chg_lab4,NULL);
XtAddCallback(bttn[5],XmNactivateCallback,chg_lab5,NULL);
XtAddCallback(bttn[6],XmNactivateCallback,chg_lab6,NULL);
```

```
XtAddCallback(bttn[7],XmNactivateCallback,chg_lab7,NULL);
XtAddCallback(bttn[8],XmNactivateCallback,chg_lab8,NULL);
XtAddCallback(bttn[9],XmNactivateCallback,chg_lab9,NULL);
XtAddCallback(bttn[10],XmNactivateCallback,chg_lab10,NULL);
XtAddCallback(bttn[11],XmNactivateCallback,chg_lab11,NULL);
XtAddCallback(bttn[12],XmNactivateCallback,chg_lab12,NULL);
XtAddCallback(bttn[13],XmNactivateCallback,chg_lab13,NULL);
XtAddCallback(bttn[14],XmNactivateCallback,chg_lab14,NULL);
XtAddCallback(bttn[15],XmNactivateCallback,chg_lab15,NULL);
XtAddCallback(bttn[16],XmNactivateCallback,chg_lab16,NULL);
XtAddCallback(bttn[17],XmNactivateCallback,chg_lab17,NULL);
XtAddCallback(bttn[18],XmNactivateCallback,chg_lab18,NULL);
XtAddCallback(bttn[19],XmNactivateCallback,chg_lab19,NULL);
XtAddCallback(bttn[20],XmNactivateCallback,chg_lab20,NULL);
XtAddCallback(bttn[21],XmNactivateCallback,chg_lab21,NULL);
XtAddCallback(bttn[22],XmNactivateCallback,chg_lab22,NULL);
XtAddCallback(bttn[23],XmNactivateCallback,chg_lab23,NULL);
XtAddCallback(bttn[24],XmNactivateCallback,chg_lab24,NULL);
XtAddCallback(bttn[25],XmNactivateCallback,chg_lab25,NULL);
XtAddCallback(bttn[26],XmNactivateCallback,chg_lab26,NULL);
XtAddCallback(bttn[27],XmNactivateCallback,chg_lab27,NULL);
XtAddCallback(bttn[28],XmNactivateCallback,chg_lab28,NULL);
XtAddCallback(bttn[29],XmNactivateCallback,chg_lab29,NULL);
XtAddCallback(bttn[30],XmNactivateCallback,chg_lab30,NULL);
XtAddCallback(bttn[31],XmNactivateCallback,chg_lab31,NULL);
XtAddCallback(bttn[32],XmNactivateCallback,chg_lab32,NULL);
XtAddCallback(bttn[33],XmNactivateCallback,chg_lab33,NULL);
XtAddCallback(bttn[34],XmNactivateCallback,chg_lab34,NULL);
XtAddCallback(bttn[35],XmNactivateCallback,chg_lab35,NULL);
XtAddCallback(bttn[36],XmNactivateCallback,chg_lab36,NULL);
XtAddCallback(bttn[37],XmNactivateCallback,chg_lab37,NULL);
XtAddCallback(bttn[38],XmNactivateCallback,chg_lab38,NULL);
XtAddCallback(bttn[39],XmNactivateCallback,chg_lab39,NULL);
XtAddCallback(bttn[40],XmNactivateCallback,chg_lab40,NULL);
XtAddCallback(bttn[41],XmNactivateCallback,chg_lab41,NULL);
XtAddCallback(bttn[42],XmNactivateCallback,chg_lab42,NULL);
XtAddCallback(bttn[43],XmNactivateCallback,chg_lab43,NULL);
XtAddCallback(bttn[44],XmNactivateCallback,chg_lab44,NULL);
XtAddCallback(bttn[45],XmNactivateCallback,chg_lab45,NULL);
XtAddCallback(bttn[46],XmNactivateCallback,chg_lab46,NULL);
XtAddCallback(bttn[47],XmNactivateCallback,chg_lab47,NULL);
XtAddCallback(bttn[48],XmNactivateCallback,chg_lab48,NULL);
                                        /* Create frame around form widget. */
frame_tm = XtVaCreateManagedWidget("frame",
xmFrameWidgetClass, form,               /* Parent. */
XmNshadowType, XmSHADOW_ETCHED_OUT,     /* Create shadow around the frame. */
                                        /* Attatche positions of frame widget with the form.
                                        */
XmNtopAttachment, XmATTACH_POSITION,XmNtopPosition, 16,
XmNbottomAttachment, XmATTACH_POSITION,XmNbottomPosition, 37,
XmNleftAttachment, XmATTACH_POSITION,XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION,XmNrightPosition, 41,NULL);

                                        /* Create frame widget for three buttons at the
                                        bottom of window. */
frame_lb = XtVaCreateManagedWidget("frame",
xmFrameWidgetClass, form,               /* Parent. */
XmNshadowType, XmSHADOW_ETCHED_OUT,     /* Create shadow around the frame. */
/* Attatche positions of frame widget with the form. */
XmNtopAttachment, XmATTACH_POSITION,XmNtopPosition, 37,
XmNbottomAttachment, XmATTACH_POSITION,XmNbottomPosition, 42,
XmNleftAttachment, XmATTACH_POSITION,XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION,XmNrightPosition, 41,NULL);
                                        /* Create form widget to place bottom three push
                                        buttons. */
form5 = XtVaCreateWidget("m_win",
xmFormWidgetClass,  frame_lb,           /* Parent. */
XmNfractionBase, 40,
XmNwidth,10,XmNheight, 10,NULL);
                                        /* Initialize the system counter structure. */
Counter[0].Line=0;                      /* Initial screen line number. */
Counter[1].Line=0;
Counter[2].Line=0;
Counter[0].Val_pos=1;                   /* Initial screen data values. */
Counter[1].Val_pos=2;
Counter[2].Val_pos=3;
                                        /* Create information label about bottom three
                                        buttons. */
XtVaCreateManagedWidget("Click above buttons to display history graph for selected system
counter.",
```

```
xmLabelWidgetClass,form5,                    /* Parent Widget */
                                             /* Attatche positions of label widget with the form.
                                             */
XmNtopAttachment, XmATTACH_POSITION,XmNtopPosition, 32,
XmNbottomAttachment, XmATTACH_POSITION,XmNbottomPosition, 4 ,
XmNleftAttachment, XmATTACH_POSITION,XmNleftPosition, 9,
XmNrightAttachment, XmATTACH_POSITION,XmNrightPosition, 39,
XmNalignment,XmALIGNMENT_BEGINNING,NULL);

                                             /* Label string for firs' default system counter. */
strcpy(Counter[0].Label,"Total User Time");
label_string=XmStringCreateSimple(Counter[0].Label);
                                             /* Create first default push button for system
                                             counter and garph. */
disp_bt1=XtVaCreateManagedWidget ("History",
xmPushButtonWidgetClass, form5,              /* Parent widget. */
XmNlabelString, label_string,                /* Label string for the push button. */
                                             /* Attatche positions of push button widget with the
                                             form. */
XmNtopAttachment, XmATTACH_POSITION,XmNtopPosition, 10,
XmNbottomAttachment, XmATTACH_POSITION,XmNbottomPosition, 29,
XmNleftAttachment, XmATTACH_POSITION,XmNleftPosition, 3,
XmNrightAttachment, XmATTACH_POSITION,XmNrightPosition, 11,NULL);
                                             /* Set the background color of the push button to
                                             green. */
XtVaSetValues(disp_bt1,XtVaTypedArg,XmNbackground,XmRString,"green",6,NULL);
                                             /* Set the user data of the push button. */
XtVaSetValues(disp_bt1,XmNuserData,0,NULL);
                                             /* Set the call back routine for the push button.
                                             Call back routine is show_stat. */
XtAddCallback(disp_bt1,XmNactivateCallback,show_stat,NULL);
                                             /* Label string for 2nd default system counter. */
strcpy(Counter[1].Label,"Dirty Page Recs");
label_string=XmStringCreateSimple(Counter[1].Label);
                                             /* Create 2nd default push button for system counter
                                             and garph. */
disp_bt2=XtVaCreateManagedWidget ("History",
xmPushButtonWidgetClass, form5,              /* Parent widget. */
XmNlabelString, label_string,                /* Label string for the push button. */
                                             /* Attatche positions of push button widget with the
                                             form. */
XmNtopAttachment, XmATTACH_POSITION,XmNtopPosition, 10,
XmNbottomAttachment, XmATTACH_POSITION,XmNbottomPosition, 29,
XmNleftAttachment, XmATTACH_POSITION,XmNleftPosition, 16,
XmNrightAttachment, XmATTACH_POSITION,XmNrightPosition, 24,NULL);
                                             /* Set the background color of the push button to
                                             green. */
XtVaSetValues(disp_bt2,XtVaTypedArg,XmNbackground,XmRString,"green",6,NULL);
                                             /* Set the user data of the push button. */
XtVaSetValues(disp_bt2,XmNuserData,1,NULL);
                                             /* Set the call back routine for the push button.
                                             Call back routine is show_stat. */
XtAddCallback(disp_bt2,XmNactivateCallback,show_stat,NULL);

                                             /* Label string for 3rd default system counter. */
strcpy(Counter[2].Label,"Semaphore Ops");
label_string=XmStringCreateSimple(Counter[2].Label);
                                             /* Create 3rd default push button for system counter
                                             and garph. */
disp_bt3=XtVaCreateManagedWidget ("History",
xmPushButtonWidgetClass, form5,              /* parent widget. */
XmNlabelString, label_string,                /* Label string. */
XmNtopAttachment, XmATTACH_POSITION,XmNtopPosition, 10,
XmNbottomAttachment, XmATTACH_POSITION,XmNbottomPosition, 29,
XmNleftAttachment, XmATTACH_POSITION,XmNleftPosition, 29,
XmNrightAttachment, XmATTACH_POSITION,XmNrightPosition, 37,NULL);
                                             /* Free up the unused string. */
XmStringFree(label_string);
                                             /* Set the background color of the push button t
                                             green. */
XtVaSetValues(disp_bt3,XtVaTypedArg,XmNbackground,XmRString,"green",6,NULL);

                                             /* Set the user data of the push button. */
XtVaSetValues(disp_bt3,XmNuserData,2,NULL);
                                             /* Set the call back routine for the push button.
                                             Call back routine is show_stat. */
XtAddCallback(disp_bt3,XmNactivateCallback,show_stat,NULL);
                                             /* Create form widget to contain scale widgets t
                                             display system counter values. */
form6 = XtVaCreateWidget("m_win",
```

```
xmFormWidgetClass,  frame_tm,
XmNfractionBase, 110,
XmNwidth,200, XmNheight, 10, NULL);
                                        /* Create first frame widget to contain first scale
                                        widget. */
frame_sc1 = XtVaCreateManagedWidget("frame",
xmFrameWidgetClass, form6,              /* Parent */
XmNshadowType, XmSHADOW_ETCHED_OUT,     /* Shadow type for frame widget. */
                                        /* Attatche positions of this widget with the parent
                                        widget. *
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 110,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 37,
XmNborderWidth, 4,                      /* Width of frame's border. */
NULL);

                                        /* Create 2nd frame widget to contain 2nd scale
                                        widget. */
frame_sc2 = XtVaCreateManagedWidget("frame",
xmFrameWidgetClass, form6,              /* Parent. */
XmNshadowType, XmSHADOW_ETCHED_IN,      /* Shadow type for frame widget. */
/* Attatche positions of this widget with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 110,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 37,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 74, NULL);
                                        /* Create 3rd frame widget to contain 3rd scale
                                        widget. */
frame_sc3 = XtVaCreateManagedWidget("frame",
xmFrameWidgetClass, form6,              /* Parent */
XmNshadowType, XmSHADOW_ETCHED_IN,      /* Shadow type for frame widget. */
/* Attatche positions of this widget with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 110,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 74,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 110,
XmNborderWidth, 4,                      /* Width of frame's border. */
NULL);
                                        /* Create form widget for first scale widget. */
fr1 = XtVaCreateManagedWidget("m_win",
xmFormWidgetClass,  frame_sc1,          /* Parent. */
XmNfractionBase, 50,
XmNwidth,100, XmNheight, 50,            /* Size of widget. */
NULL);
                                        /* Create form widget for 2nd scale widget. */
fr2 = XtVaCreateManagedWidget("m_win",
xmFormWidgetClass,  frame_sc2,          /* Parent. */
XmNfractionBase, 50,
XmNwidth,100, XmNheight, 50,            /* Size of widget. */
NULL);
                                        /* Create form widget for 3rd scale widget. */
fr3 = XtVaCreateManagedWidget("m_win",
xmFormWidgetClass,  frame_sc3,          /* Parent. */
XmNfractionBase, 50,
XmNwidth,100, XmNheight, 50,            /* Size of widget. */
NULL);
                                        /* Create first system counter scale widget and
                                        attach it to the form 'fr1'. */
scale_ct[1] = XtVaCreateManagedWidget ("scale",
xmScaleWidgetClass, fr1,                /* Parent. */
XtVaTypedArg, XmNtitleString, XmRString, "",3, /* Scale name. */
XmNmaximum,        2000,                /* Maximun default value of scale widget. */
XmNminimum,        0,                   /* Minimum default value of scale widget. */
XmNvalue,          0,                   /* Initial value of scale widget. */
XmNshowValue,      True,                /* Value display on the scale widget. */
XmNscaleWidth, 25,                      /* Scale width. */
                                        /* Attatche positions of this scale with the parent
                                        widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 50,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 27,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 40, NULL);
                                        /* Create label for maximum value for scale widget.
                                        */
XtVaCreateManagedWidget("Max:",
xmLabelWidgetClass, fr1,                /* Parent. */
                                        /* Attatche positions of this label with the parent
                                        widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
```

```
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 4,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 6, NULL);

                                        /* Create label for maximum default value. */
label_string_Max1=XmStringCreateSimple("2000");
lb_m1=XtVaCreateManagedWidget("max",
xmLabelWidgetClass, fr1,                /* Parent. */
XmNlabelString, label_string_Max1,      /* String for label. */
                                        /* Attatche positions of this label with the parent
                                        widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 4,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 7,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 14, NULL);
                                        /* Free up the motif string because it is no longer
                                        in use. */
XmStringFree(label_string_Max1);
                                        /* Create label for minimum value for scale widget.
                                        */
XtVaCreateManagedWidget("Min:",
xmLabelWidgetClass, fr1, /* Parent. */
/* Attatche positions of this label with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 46,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 50,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 6, NULL);
                                        /* Create label for minimum default value. */
label_string_Min1=XmStringCreateSimple("0");
XtVaCreateManagedWidget("min",
xmLabelWidgetClass, fr1,                /* Parent. */
XmNlabelString, label_string_Min1,      /* String for label. */
/* Attatche positions of this label with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 46,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 50,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 7,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 14, NULL);
                                        /* Free up the motif string because it is no longer
                                        in use. */
XmStringFree(label_string_Min1);

                                        /* Create 2nd system counter scale widget and attach
                                        it to the form 'fr2'. */
scale_ct[3] = XtVaCreateManagedWidget ("scale", xmScaleWidgetClass, fr2,
XtVaTypedArg, XmNtitleString, XmRString, "",3, /* Scale name. */
XmNmaximum,         2000,               /* Maximun default value of scale widget. */
XmNminimum,         0,                  /* Minimum default value of scale widget. */
XmNvalue,           0,                  /* Initial value of scale widget. */
XmNshowValue,       True,               /* Value display on the scale widget. */
XmNscaleWidth, 25,                      /* Scale width. */
/* Attatche positions of this scale with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 50,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 25,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 40, NULL);
                                        /* Create label for maximum value for scale widget.
                                        */
XtVaCreateManagedWidget("Max:",
xmLabelWidgetClass, fr2,                /* Parent. */
                                        /* Attatche positions of this label with the parent
                                        widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 4,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 6, NULL);
                                /* Create label for maximum default value. */
label_string_Max2=XmStringCreateSimple("2000");
lb_m2=XtVaCreateManagedWidget("max",
xmLabelWidgetClass, fr2,                /* Parent. */
XmNlabelString, label_string_Max2,      /* String for label. */
                                        /* Attatche positions of this label with the parent
                                        widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 4,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 7,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 14, NULL);
                                        /* Free up the motif string because it is no longer
                                        in use. */
XmStringFree(label_string_Max2);
```

```
                                        /* Create label for minimum value for scale widget.
                                        */
XtVaCreateManagedWidget("Min:",
xmLabelWidgetClass, fr2,                /* Parent. */
                                        /* Attatche positions of this label with the parent
                                        widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 46,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 50,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 6, NULL);

                                        /* Create label for minimum default value. */
label_string_Min2=XmStringCreateSimple("0");
XtVaCreateManagedWidget("min",
xmLabelWidgetClass, fr2,                /* Parent. */
XmNlabelString, label_string_Min2,      /* String for label. */
/* Attatche positions of this label with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 46,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 50,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 7,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 14, NULL);

                                        /* Free up the motif string because it is no longer
                                        in use. */
XmStringFree(label_string_Min2);

                                        /* Create 3rd system counter scale widget and attach
                                        it to the form 'fr3' */
scale_ct[5] = XtVaCreateManagedWidget ("scale", xmScaleWidgetClass, fr3,
XtVaTypedArg, XmNtitleString, XmRString, "",3, /* Scale name. */
XmNmaximum,       2000,                 /* Maximun default value of scale widget. */
XmNminimum,       0,                    /* Minimum default value of scale widget. */
XmNvalue,         0,                    /* Initial value of scale widget. */
XmNshowValue,     True,                 /* Value display on the scale widget. */
XmNscaleWidth, 25,                      /* Scale width. */
/* Attatche positions of this scale with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 50,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 25,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 40, NULL);

                                        /* Create label for maximum value for scale widget.
                                        */
XtVaCreateManagedWidget("Max:",
xmLabelWidgetClass, fr3,                /* Parent. */
/* Attatche positions of this label with the parent widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 4,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 6, NULL);

                                        /* Create label for maximum default value. */
label_string_Max3=XmStringCreateSimple("2000");
lb_m3=XtVaCreateManagedWidget("max",
xmLabelWidgetClass, fr3,                /* Parent. */
XmNlabelString, label_string_Max3,      /* String for label. */
                                        /* Attatche positions of this label with the parent
                                        widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 4,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 7,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 14, NULL);
                                        /* Free up the motif string because it is no longer
                                        in use. */
XmStringFree(label_string_Max3);
                                        /* Create label for minimum value for scale widget.
                                        */
XtVaCreateManagedWidget("Min:",
xmLabelWidgetClass, fr3,                /* Parent. */
                                        /* Attatche positions of this label with the parent
                                        widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 46,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 50,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 6, NULL);

                                        /* Create label for minimum default value. */
label_string_Min3=XmStringCreateSimple("0");
XtVaCreateManagedWidget("min",
xmLabelWidgetClass, fr3,                /* Parent. */
```

```
XmNlabelString, label_string_Min3,    /* String for label. */
                                      /* Attatche positions of this label with the parent
                                      widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 46,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 50,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 7,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 14, NULL);
                                      /* Free up the motif string because it is no longer
                                      in use. */
XmStringFree(label_string_Min3);

                                      /* Manage all widgets to make them appear on the
                                      screen. */
XtManageChild(form6);
XtManageChild(form5);
XtManageChild(form);
while(1){                             /* Wait for the user tp press the 'SYSTEM COUNTERS'
                                      push button. */
        read(counter_pipe[0],&counter_start,sizeof(counter_start));
        if(counter_start==25)         /* If user presses the 'SYSTEM COUNTERS' button then
                                      exit. */
                break;
        }
                                      /* Realize the root window to make it appears on the
                                      screen. */
XtRealizeWidget (toplevel_ctr);
GetData=4;                            /* Control word. */
                                      /* Send the signal to the child process to start
                                      gathering data about system counters. */
write(pipe_getdata[1],&GetData,sizeof(GetData));
for(;;){                              /* This loop waits for X events and then process
                                      accordingly. */
        if(flag_cursor==0){           /* If the cursor is in normal shape. */
                flag_cursor=1;
                                      /* Set cursor to watch shape. */
        cursor=XCreateFontCursor(dpy_ctr,XC_watch);
        attrs.cursor=cursor;
        XChangeWindowAttributes(dpy_ctr,XtWindow(toplevel_ctr),CWCursor,&attrs);
        )

                                      /* Update the system counter window by sending
                                      exposure events to the X server. */
        XmUpdateDisplay(toplevel_ctr);
                                      /* Select the interested X events. Interested events
                                      are mouse button press and release and exposure
                                      events. */
        if(XCheckMaskEvent(dpy_ctr,ButtonPressMask | ButtonReleaseMask |
        ExposureMask,&event))
                                      /* Dispatch the interested events to the X server.
                                      */
        {       XtDispatchEvent(&event); }
                                      /* Display the system counters values on the scale
                                      widgets. */

        show_graph();

        if(flag_cursor==1){           /* If the cursor is in watch shape. */
        flag_cursor=2;
                                      /* Set cursor back to normal shape. */

        attrs.cursor=None;
        XChangeWindowAttributes(dpy_ctr,XtWindow(toplevel_ctr),CWCursor,&attrs);
        }
}                                     /* End main events loop. */
}                                     /* Parent process. */
}                                     /* child3 ends. */
                                      /* This routine reads system counters data from
                                      pipe, send by child process and displays on the
                                      three scale widgets of the system counters window.
                                      */
void show_graph()
{
#define SET     1
int flag=0;
int match1,match2,match3;
static int fg_counter=0;
int x_current,y_current,y_flag;
int x_previous,y_previous;
int height_current,width_current;
int height_previous,width_previous;
int read_line,temp;
int max,min,my_value;
```

```
int read_first_value,read_2nd_value,read_3rd_value;/* Read values from pipe. */
char cur_st[7];
char str_max[9];
float x1,y1;
                                        /* Default maximum values for system's counter. */
static int max_ctr1_val=2000;
static int max_ctr2_val=2000;
static int max_ctr3_val=2000;
GC gc;
read_first_value=read_2nd_value=read_3rd_value=0;
max=min=my_value=0;
read_line=0;
match1=match2=match3=0;
x1=20.0;                                /*If system counter's value acceed from 2000 then
                                        the default value is increases by 20 % of the maximum
                                        value. */
y1=100.0;

                                        /* Read screen line. */
read(pipe_cntrB[0],&read_line,sizeof(read_line));
                                        /* Read first value from the screen. */
read(pipe_cntrB[0],&read_first_value,sizeof(read_first_value));
                                        /* Read 2nd value from the screen. */
read(pipe_cntrB[0],&read_2nd_value,sizeof(read_2nd_value));
                                        /* Store first and second values. */
Prev[read_line].P1st=read_first_value;
Prev[read_line].P2nd=read_2nd_value;
                                        /* Store first and second values for the graph. */
store_it(read_line,read_first_value,read_2nd_value);

if(input_select==0)
{                                       /* Select the X events for system counter window. */
input_select=1;
XSelectInput(dpy_ctr,XtWindow(toplevel_ctr),ButtonPressMask|
ButtonReleaseMask|ExposureMask);
}
if(read_line==0)
{                                       /* If line is zero then read 3rd value from curses
                                        screen and then store it. */
read(pipe_cntrB[0],&read_3rd_value,sizeof(read_3rd_value));
Prev[read_line].P3rd=read_3rd_value;
}
                                        /* If the read screen line is matched to the first
                                        scale widget's structure value then set the raed
                                        value to the first scale value. */
if(read_line==Counter[0].Line)
{
if(Counter[0].Val_pos==1)               /* If the value is on position 1st on the curses
                                        screen. */
my_value=read_first_value;              /* Use first value. */
else if(Counter[0].Val_pos==2)          /* If the value is on position 2nd on the curses
                                        screen. */
my_value=read_2nd_value;                /* Use 2nd value. */
else if(Counter[0].Val_pos==3)          /* If the value is on position 3rd on the curses
                                        screen. */
my_value=read_3rd_value;                /* Use 3rd value. */

if( (my_value <= 2000)&&(max_ctr1_val != 2000) )
                                        /* Check the maximum value is in default range. */
{
max_ctr1_val=max=2000;                  /* If yes then assign default maximum value. */

                                        /* Change the Maximum value for first scale widget.
                                        */
sprintf(str_max,"%d",max);
XtVaSetValues(lb_m1,XmNlabelString, XmStringCreateSimple(str_max),NULL);
                                        /* Set the maximum and minimum values on the scale
                                        widget. */
XtVaSetValues(scale_ct[1],
XmNmaximum,     max,                    /* Maximum value. */
XmNminimum,     min,                    /* Minimum value. */
XmNvalue,       min,                    /* Minimum value. */
NULL);
}
                                        /* If the counter value is greater than the default
                                        maximum value then add the 20% of the actual value
                                        to the read counter value. */
else if(my_value > max_ctr1_val)
{
max_ctr1_val=( (x1/y1)*my_value )+my_value; /* Adjust the maximum range. */
max=max_ctr1_val;
```

```
                                        /* Change the Maximum value for first scale widget.
                                        */
sprintf(str_max,"%d",max);
XtVaSetValues(lb_m1,XmNlabelString, XmStringCreateSimple(str_max),NULL);

                                        /* Set the maximum and minimum values on the scale
                                        widget. */
XtVaSetValues(scale_ct[1],
XmNmaximum,     max,                    /* Maximum value. */
XmNminimum,     min,                    /* Minimum value. */
XmNvalue,       min,                    /* Minimum value. */
NULL);
}
                                        /* Display the actual value on the scale widget. */
XmScaleSetValue(scale_ct[1],my_value);
match1=1;                               /* First value has been displayed. */
}                                       /* counter 0 ends. */
                                        /* If the read screen line is matched to the 2nd
                                        scale widget's structure value then set read value
                                        to 2nd scale value. */
if(read_line==Counter[1].Line)
{
if(Counter[1].Val_pos==1)               /* If the value is on position 1st on the curses
                                        screen. */
my_value=read_first_value;              /* Use first value. */
else if(Counter[1].Val_pos==2)          /* If the value is on position 2nd on the curses
                                        screen. */
my_value=read_2nd_value;                /* Use 2nd value. */
else if(Counter[1].Val_pos==3)          /* If the value is on position 3rd on the curses
                                        screen. */
my_value=read_3rd_value;                /* Use 3rd value. */
if( (my_value <= 2000)&&(max_ctr2_val != 2000) )
{                                       /* Check the maximum value is in default range. */
max_ctr2_val=max=2000;                  /* If yes then assign default maximum value. */

                                        /* Change the Maximum value for first scale widget.
                                        */
sprintf(str_max,"%d",max);
XtVaSetValues(lb_m2,XmNlabelString, XmStringCreateSimple(str_max),NULL);

                                        /* Set the maximum and minimum values on the scale
                                        widget. */
XtVaSetValues(scale_ct[3],
XmNmaximum,     max,                    /* Maximum value. */
XmNminimum,     min,                    /* Minimum value. */
XmNvalue,       min,                    /* Minimum value. */
NULL);
}
                                        /* If the counter value is greater than the default
                                        maximum value then add the 20% of the actual value
                                        to the read counter value. */
else if(my_value > max_ctr2_val)
{
max_ctr2_val=( (x1/y1)*my_value )+my_value; /* Adjust the maximum range. */
max=max_ctr2_val;
                                        /* Change the Maximum value for first scale widget.
                                        */
sprintf(str_max,"%d",max);
XtVaSetValues(lb_m2,XmNlabelString, XmStringCreateSimple(str_max),NULL);
                                        /* Set the maximum and minimum values on the scale
                                        widget. */
XtVaSetValues(scale_ct[3],
XmNmaximum,     max,                    /* Maximum value. */
XmNminimum,     min,                    /* Minimum value. */
XmNvalue,       min,                    /* Minimum value. */
NULL);
}
                                        /* Display the actual value on the scale widget. */
XmScaleSetValue(scale_ct[3],my_value);
match2=1;                               /* Second value has been displayed. */
}                                       /* counter 1 ends */

                                        /* If the read screen line is matched to the 3rd
                                        scale widget's structure value then set the read
                                        value to the 3rd scale widget. */
if(read_line==Counter[2].Line)
{
if(Counter[2].Val_pos==1)               /* If the value is on position 1st on the curses
                                        screen. */
my_value=read_first_value;              /* Use first value. */
```

```
else if(Counter[2].Val_pos==2)          /* If the value is on position 2nd on the curses
                                        screen. */
my_value=read_2nd_value;                /* Use 2nd value. */
else if(Counter[2].Val_pos==3)          /* If the value is on position 3rd on the curses
                                        screen. */
my_value=read_3rd_value;                /* Use 3rd value. */

if( (my_value <= 2000)&&(max_ctr3_val != 2000) )
{                                       /* Check the maximum value is in default range. */
max_ctr3_val=max=2000;                  /* If yes then assign default maximum value. */

                                        /* Change the Maximum value for first scale widget.
                                        */
sprintf(str_max,"%d",max);
XtVaSetValues(lb_m3,XmNlabelString, XmStringCreateSimple(str_max),NULL);
                                        /* Set the maximum and minimum values on the scale
                                        widget. */
XtVaSetValues(scale_ct[5],
XmNmaximum,     max,                    /* Maximum value. */
XmNminimum,     min,                    /* Minimum value. */
XmNvalue,       min,                    /* Minimum value. */
NULL);
}

                                        /* If the counter value is greater than the default
                                        maximum value then add the 20% of the actual value
                                        to the read counter value. */
else if(my_value > max_ctr3_val)
{
max_ctr3_val=( (x1/y1)*my_value )+my_value; /* Adjust the maximum range. */
max=max_ctr3_val;

                                        /* Change the Maximum value for first scale widget.
                                        */
sprintf(str_max,"%d",max);
XtVaSetValues(lb_m3,XmNlabelString, XmStringCreateSimple(str_max),NULL);
                                        /* Set the maximum and minimum values on the scale
                                        widget. */
XtVaSetValues(scale_ct[5],              /* set the scale. */
XmNmaximum,     max,                    /* Maximum value. */
XmNminimum,     min,                    /* Minimum value. */
XmNvalue,       min,                    /* Minimum value. */
NULL);
}
                                        /* Display the actual value on the scale widget. */
XmScaleSetValue(scale_ct[5],my_value);
match3=1;                               /* Third value has been displayed. */
}                                       /* counter 2 ends. */
                                        /* If selected line does not appear on the curses
                                        screen then use the stored values for system
                                        counters. */
if(!match1)                             /* For first scale widget. */
{
if(Counter[0].Val_pos==1)               /* If the value is on position 1st on the curses
                                        screen. */
my_value=Prev[Counter[0].Line].PFirst; /* Use first stored value. */
else if(Counter[0].Val_pos==2) /* If the value is on position 2nd on the curses screen. */
my_value=Prev[Counter[0].Line].P2nd; /* Use 2nd stored value. */
else if(Counter[0].Val_pos==3)          /* If the value is on position 3rd on the curses
                                        screen. */
my_value=Prev[Counter[0].Line].P3rd; /* Use 3rd stored value. */
if( (my_value <= 2000)&&(max_ctr1_val != 2000) )
{                                       /* Check the maximum value is in default range. */
max_ctr1_val=max=2000;                  /* If yes then assign default maximum value. */
                                        /* Change the Maximum value for first scale widget.
                                        */
sprintf(str_max,"%d",max);
XtVaSetValues(lb_m1,XmNlabelString, XmStringCreateSimple(str_max),NULL);

                                        /* Set the maximum and minimum values on the scale
                                        widget. */
XtVaSetValues(scale_ct[1],
XmNmaximum,     max,                    /* Maximum value. */
XmNminimum,     min,                    /* Minimum value. */
XmNvalue,       min,                    /* Minimum value. */
NULL);
}

                                        /* If the counter value is greater than the default
                                        maximum value then add the 20% of the actual value
                                        to the read counter value. */
else if(my_value > max_ctr1_val)
```

```
{
max_ctrl_val=( (x1/y1)*my_value )-my_value; /* Adjust the maximum range. */
max=max_ctrl_val;

                                        /* Change the Maximum value for first scale widget.
                                        */
sprintf(str_max,"%d",max);
XtVaSetValues(lb_m1,XmNlabelString, XmStringCreateSimple(str_max),NULL);
                                        /* Set the maximum and minimum values on the scale
                                        widget. */
XtVaSetValues(scale_ct[1],
XmNmaximum,     max,                    /* Maximum value. */
XmNminimum,     min,                    /* Minimum value. */
XmNvalue,       min,                    /* Minimum value. */
NULL);
}
                                        /* Display the actual value on the scale widget. */
XmScaleSetValue(scale_ct[1],my_value);
}

                                        /* If selected line does not appear on the curses
                                        screen then use the stored values for system
                                        counters. */
if(!match2)                             /* For 2nd scale widget. */
{
if(Counter[1].Val_pos==1)               /* If the value is on position 1st on the curses
                                        screen. */
my_value=Prev[Counter[1].Line].PFirst; /* Use first stored value. */
else if(Counter[1].Val_pos==2) /* If the value is on position 2nd on the curses screen. */
my_value=Prev[Counter[1].Line].P2nd; /* Use 2nd stored value. */
else if(Counter[1].Val_pos==3)          /* If the value is on position 3rd on the curses
                                        screen. */
my_value=Prev[Counter[1].Line].P3rd; /* Use 3rd stored value. */
if( (my_value <= 2000)&&(max_ctr2_val != 2000) )
{                                       /* Check the maximum value is in default range. */
max_ctr2_val=max=2000;                  /* If yes then assign default maximum value. */

                                        /* Change the Maximum value for first scale widget.
                                        */
sprintf(str_max,"%d",max);
XtVaSetValues(lb_m2,XmNlabelString, XmStringCreateSimple(str_max),NULL);
                                        /* Set the maximum and minimum values on the scale
                                        widget. */
XtVaSetValues(scale_ct[3],
XmNmaximum,     max,                    /* Maximum value. */
XmNminimum,     min,                    /* Minimum value. */
XmNvalue,       min,                    /* Minimum value. */
NULL);
}
                                        /* If the counter value is greater than the default
                                        maximum value then add the 20% of the actual value
                                        to the read counter value. */
else if(my_value > max_ctr2_val)
{
max_ctr2_val=( (x1/y1)*my_value )+my_value; /* Adjust the maximum range. */
max=max_ctr2_val;

                                        /* Change the Maximum value for first scale widget.
                                        */
sprintf(str_max,"%d",max);
XtVaSetValues(lb_m2,XmNlabelString, XmStringCreateSimple(str_max),NULL);
                                        /* Set the maximum and minimum values on the scale
                                        widget. */
XtVaSetValues(scale_ct[3],
XmNmaximum,     max,                    /* Maximum value. */
XmNminimum,     min,                    /* Minimum value. */
XmNvalue,       min,                    /* Minimum value. */
NULL);
}
                                        /* Display the actual value on the scale widget. */
XmScaleSetValue(scale_ct[3],my_value);
}
                                        /* If selected line does not appear on the curses
                                        screen then use the stored values for system
                                        counters. */
if(!match3)                             /* For 3rd scale widget. */
{
if(Counter[2].Val_pos==1)
my_value=Prev[Counter[2].Line].PFirst;
else if(Counter[2].Val_pos==2)
```

```
my_value=Prev[Counter[2].Line].P2nd;
else if(Counter[2].Val_pos==3)
my_value=Prev[Counter[2].Line].P3rd;


if( (my_value <= 2000)&&(max_ctr3_val != 2000) )
{
max_ctr3_val=max=2000;
                                        /* Change the Max: label for scale 1. */
sprintf(str_max,"%d",max);
XtVaSetValues(lb_m3,XmNlabelString,
XmStringCreateSimple(str_max),NULL);
XtVaSetValues(scale_ct[5],              /* Set the scale. */
XmNmaximum,     max,
XmNminimum,     min,                    /* Zero. */
XmNvalue,       min,
NULL);
}
else if(my_value > max_ctr3_val)      /* Adjust the range. */
{
max_ctr3_val=( (xl/yl)*my_value )+my_value; /* 20% */
max=max_ctr3_val;
                                        /* Change the Max: label for scale 1. */
sprintf(str_max,"%d",max);
XtVaSetValues(lb_m3,XmNlabelString,
XmStringCreateSimple(str_max),NULL);

XtVaSetValues(scale_ct[5],              /* Set the scale. */
XmNmaximum,     max,
XmNminimum,     min,                    /* Zero. */
XmNvalue,       min,
NULL);
}
XmScaleSetValue(scale_ct[5],my_value);
}

}                                       /* end show graph. */
                                        /* This routine creates a pop up display for system
                                        counters history. */
void
show_stat(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
Cursor cursor;                          /* Cursor structure. */
XSetWindowAttributes attrs;
Display *dpy_wch=XtDisplay(toplevel_ctr);/* Get display ID of root window. */

XGCValues gcv;                          /* Graphics context values. */
GC gc;
Widget  form;
Arg args[2];
char temp_str[]="History for ";         /* Starting part of title string. */
XmString label_str;
int ct_no;
Widget dialog,draw,frame_pop1,frame_pop2,bttn_down;
char buf[25]="System Counter History"; /* Window title. */
                                        /* Change the cursor to busy state. */
cursor=XCreateFontCursor(dpy_wch,XC_watch);
attrs.cursor=cursor;
XChangeWindowAttributes(dpy_wch,XtWindow(toplevel_ctr),CWCursor,&attrs);
                                        /* Get the user date from the parent widget. The
                                        user date contain the counter number. */
XtVaGetValues(w,XmNuserData,&ct_no,NULL);
which_ctr=ct_no;                        /* Store the counter number. */
                                        /* Create the pop up widget. */
dialog = XtVaCreatePopupShell ("popup",
xmDialogShellWidgetClass, GetTopShell(w), /* Parent. */
XmNtitle,               buf,            /* Title string */
XmNallowShellResize, True,              /* Allow the user to resize the pop up window. */
XmNdeleteResponse,   XmDESTROY,         /* Allow the user to destroy the pop up shell. */
XmNx,       200, XmNy,       200,       /* Size of pop up window. */
NULL);
                                        /* Create the form widget. */
form=XtVaCreateWidget("main_window",
xmFormWidgetClass,  dialog,
XmNfractionBase, 40,
XmNwidth, 750, XmNheight, 400, NULL);
                                        /* Create the frame widget. */
```

```
frame_pop1 = XtVaCreateManagedWidget("frame",
xmFrameWidgetClass, form, /* Parent. */
XmNshadowType, XmSHADOW_ETCHED_OUT,
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 2,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 40, NULL);
                                        /* Get the label string from selected counter push
                                        button. */
strcat(temp_str,Counter[ct_no].Label);
label_str=XmStringCreateSimple(temp_str);
                                        /* Make the title label. */
XtVaCreateManagedWidget("lb",
xmLabelWidgetClass, frame_pop1,
XmNlabelString, label_str,
NULL);
                                        /* Free up the unused string. */
XmStringFree(label_str);

                                        /* Create the frame widget. */
frame_pop2 = XtVaCreateManagedWidget("frame",
xmFrameWidgetClass, form,
XmNshadowType, XmSHADOW_ETCHED_OUT,
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 2,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 40,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 38, NULL);

                                        /* Create 'OK' push button and attach it to the
                                        form. */
bttn_down=XtVaCreateManagedWidget ("OK",
xmPushButtonWidgetClass, form, /* Parent. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 37,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 40,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 38,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 40, NULL);

                                        /* Set the color of push button. */
XtVaSetValues(bttn_down,XtVaTypedArg,XmNbackground,XmRString,"Green",7,NULL);
                                        /* Attach the dialog widget to the button's user
                                        data. */
XtVaSetValues(bttn_down, XmNuserData, dialog, NULL);
                                        /* Set call back routine for the push button. Call
                                        back routine is Popdown2. */
XtAddCallback(bttn_down,XmNactivateCallback, Popdown2,NULL);
                                        /* Create drawing area for draw the history graph.
                                        */
draw = XtVaCreateManagedWidget("draw_a",
xmDrawingAreaWidgetClass, frame_pop2,
XmNwidth, 750,XmNheight, 400,           /* Size of drawing area. */
NULL);

dpy_popup_CTR = XtDisplay(draw);        /* Get the display ID. */
cmap=DefaultColormapOfScreen(XtScreen(draw));/* Get the color of drawing area. */
scr_popup_ctr = XtScreen(draw);         /* Get the screen ID. */
                                        /* Set the drawing area as a work area of form
                                        widget. */
XtVaSetValues(form,XmNworkWindow, draw,NULL);
                                        /* Set the call back routine for drawing area
                                        widget. */
XtAddCallback(draw, XmNexposeCallback, show_utilization2, NULL);
                                        /* Get the foreground color of the drawing area. */
gcv.foreground = BlackPixelOfScreen(scr_popup_ctr);
                                        /* Set the graphics context of drawing area. */
gc = XCreateGC(dpy_popup_CTR, RootWindowOfScreen(scr_popup_ctr), GCForeground, &gcv);
                                        /* Attach the GC to the user data of drawing area.
                                        */
XtVaSetValues(draw,XmNuserData,gc,NULL);
                                        /* Manage the form. */
XtManageChild(form);
                                        /* Make the pop up window appear on the screen. */
XtPopup (dialog, XtGrabNone),
)
                                        /* This routine is executed whenever user presses
                                        the OK button on the pop up window. */
void
Popdown2(w, client_data, call_data)
Widget w;
XtPointer client_data;                  /* Client user data. */
XtPointer call_data;                    /* Routine call data. */
```

```
{
Widget wid;
int ct_no=0;

XtVaGetValues(w, XmNuserData, &wid, NULL);/* Get the parent widget of OK push button. */
XtDestroyWidget(wid);                     /* Destroy the parent widget. */
}
                                          /* This routine is used to calculate the y-axis
                                          label for the history graphics. */
void y_name(char *graph,int Line,int pos)
{
if( (Line==0&&pos==1)||(Line==1&&pos==1)||(Line==2&&pos==1) )
strcpy(graph,"% Progress");
else if( (Line==3&&pos==1)||(Line==4&&pos==1)||
        (Line==5&&pos==1)||(Line==7&&pos==1)||(Line==8&&pos==1)||
        (Line==6&&pos==1)||(Line==9&&pos==1)||(Line==10&&pos==1) )
strcpy(graph,"Process");
else if(Line==11&&pos==1)
strcpy(graph,"System Calls");
else if(Line==12&&pos==1)
strcpy(graph,"Context Switches");
else if(Line==13&&pos==1)
strcpy(graph,"Interrupts");
else if(Line==14&&pos==1)
strcpy(graph,"Traps");
else if(Line==15&&pos==1)
strcpy(graph,"Forks");
else if(Line==16&&pos==1)
strcpy(graph,"VForks");
else if(Line==17&&pos==1)
strcpy(graph,"Execs");
else if(Line==18&&pos==1)
strcpy(graph,"Disk Transf.");
else if(Line==19&&pos==1)
strcpy(graph,"# KB Transf.");
else if( (Line==20&&pos==1)||(Line==21&&pos==1) )
strcpy(graph,"# Chars.");
else if(Line==22&&pos==1)
strcpy(graph,"Page Faults");
else if( (Line==23&&pos==1)||(Line==0&&pos==2) )
strcpy(graph,"Page Reclaims");
else if((Line==1&&pos==2)||(Line==2&&pos==2)||(Line==3&&pos==2)||
(Line==4&&pos==2)||(Line==6&&pos==2)||(Line==8&&pos==2) )
strcpy(graph,"Pages");
else if( (Line==5&&pos==2)||(Line==7&&pos==2) )
strcpy(graph,"Process");
else if( (Line==9&&pos==2)||(Line==10&&pos==2)||(Line==11&&pos==2)
||(Line==12&&pos==2)||(Line==19&&pos==2)||(Line==20&&pos==2))
strcpy(graph,"Kbytes");
else if( Line==13&&pos==2)
strcpy(graph,"# Reads");
else if( Line==14&&pos==2)
strcpy(graph,"# Writes");
else if( Line==15&&pos==2)
strcpy(graph,"% FS Reads");
else if( Line==16&&pos==2)
strcpy(graph,"% FS Writes");
else if( Line==17&&pos==2)
strcpy(graph,"Raw Device Reads");
else if( Line==18&&pos==2)
strcpy(graph,"Raw Device Writes");
else if( Line==21&&pos==2)
strcpy(graph,"Locks");
else if( Line==22&&pos==2)
strcpy(graph,"% Locks");
else if( Line==23&&pos==2)
strcpy(graph,"Message Ops");
else if( Line==0&&pos==3)
strcpy(graph,"Semaphore Ops");
}
                                          /* This routine is used to draw the histroy graphics
                                          on the drawing area. */
void
show_utilization2(Widget w, XtPointer data,
         XmDrawingAreaCallbackStruct *cbk)
{
        char  str1[75],str2[75];
    GC gc;
    Window win = XtWindow(w);
        int len1,x,i,j;
```

```
        int height,base=354;            /* Height and base values of graph. */
        int C_line,Pos,Div,Point[22];
        int y[22],Local[22],temp;
        int do_again,flag_base;
        int add,div,ok=0;
        float res[24],multiply;
        char str3[20],str4[20],str5[20];
        char lab_name[70];
        XColor xcolour,spare;
        long int fill_pixel=1;
        long int fill_pixel2=1;         /* For normal black color. */
        int zero_i[23];
char Hr[20];                            /* Hour, minute, and second string for time. */
char Mn[20];
char Sc[20];
char intvl1[11];                        /* Interval for sampling for graph data. */
char intvl2[11];
char total_intvl[70]="Sampling Time: "; /* Sampling time. */
int sec,MIN,ANS,S_MIN;
Cursor cursor;
XSetWindowAttributes attrs;
Display *dpy_wch=XtDisplay(toplevel_ctr);
        int high_val,Low_val,H_i,L_i,y_low,y_high;
        H_i=L_i=y_low=y_high=high_val=Low_val=0;
        flag_base=0;
        XClearWindow(dpy_popup_CTR,win);
                                        /* Change the color of the foreground. */
        XAllocNamedColor(dpy_popup_CTR,cmap,"Red",&xcolour,&spare);
        fill_pixel=xcolour.pixel; /* Store this color. */
                                        /* Create another foreground color to draw the axis
                                        lines. */
XAllocNamedColor(dpy_popup_CTR,cmap,"Black",&xcolour,&spare);
fill_pixel2=xcolour.pixel;              /* Store this color. */
        C_line=Pos=Div=temp=0;
                                        /* Get the user data. */
        XtVaGetValues(w, XmNuserData, &gc, NULL);
        x=10;                           /* y-axis position. */
                                        /* Draw x and y-axis lines. */
        XDrawLine(dpy_popup_CTR,win,gc,x,10,x,350);
        XDrawLine(dpy_popup_CTR,win,gc,x,350,650,350);
        strcpy(str1,"^");
        len1 = strlen(str1);
        XDrawString(dpy_popup_CTR, win, gc, 6, 20, str1, len1);
        C_line=Counter[which_ctr].Line;/* Get the line number. */
        Pos=Counter[which_ctr].Val_pos;
                                        /* Get the y-axis label based on the counter number.
                                        */
        y_name(str1,C_line,Pos);        /* y-axis name. */
        strcpy(lab_name,str1);
                                        /* Determine the label to display maximum and
                                        minimum value of counter. */
        if( (C_line==0)&&(Pos==1)||(C_line==1)&&(Pos==1)||
              (C_line==2)&&(Pos==1) )
              strcpy(lab_name,"%");
        else if( (C_line==11)&&(Pos==1) )
              strcpy(lab_name,"Calls");
        else if( (C_line==12)&&(Pos==1) )
              strcpy(lab_name,"CSW");
        else if( (C_line==13)&&(Pos==1) )
              strcpy(lab_name,"Int");
        else if( (C_line==18)&&(Pos==1) )
              strcpy(lab_name,"Txfr");
        else if( (C_line==9)&&(Pos==2)||(C_line==11)&&(Pos==2)||
              (C_line==19)&&(Pos==1)||(C_line==10)&&(Pos==2)||
              (C_line==19)&&(Pos==2)||(C_line==20)&&(Pos==2)||
              (C_line==12)&&(Pos==2) )
              strcpy(lab_name,"KB");
        else if( (C_line==6)&&(Pos==1)||(C_line==3)&&(Pos==1)||
              (C_line==9)&&(Pos==1)||(C_line==10)&&(Pos==1)||
              (C_line==5)&&(Pos==1)||(C_line==4)&&(Pos==1)||
              (C_line==7)&&(Pos==1)||(C_line==5)&&(Pos==2)
             ||(C_line==7)&&(Pos==2)||(C_line==8)&&(Pos==1) )
              strcpy(lab_name,"Procs");
        else if( (C_line==20)&&(Pos==1)||(C_line==21)&&(Pos==1) )
              strcpy(lab_name,"Chars");
        else if( (C_line==22)&&(Pos==1) )
              strcpy(lab_name,"Flts");
        else if( (C_line==23)&&(Pos==1)||(C_line==0)&&(Pos==2) )
              strcpy(lab_name,"Pages");
        else if( (C_line==15)&&(Pos==2)||(C_line==16)&&(Pos==2) )
```

```
        strcpy(lab_name,"%");
else if( (C_line==17)&&(Pos==2)||(C_line==13)&&(Pos==2) )
        strcpy(lab_name,"#Rd");
else if( (C_line==18)&&(Pos==2)||(C_line==14)&&(Pos==2) )
        strcpy(lab_name,"#Wrt");
else if( (C_line==23)&&(Pos==2) )
        strcpy(lab_name,"Msg");
else if( (C_line==3)&&(Pos==3) )
        strcpy(lab_name,"Sem");
else if( (C_line==16)&&(Pos==1) )
        strcpy(lab_name,"VFrk");
else if( (C_line==22)&&(Pos==2) )
        strcpy(lab_name," Lks");
strcat(lab_name,":");
len1 = strlen(str1);
XDrawString(dpy_popup_CTR, win, gc, 17, 19, str1, len1);
strcpy(str1,">");
len1 = strlen(str1);
XDrawString(dpy_popup_CTR, win, gc, 646, 357, str1, len1);
strcpy(str1,"Time");
len1 = strlen(str1);
                                /* Print the label for x axis. */
XDrawString(dpy_popup_CTR, win, gc, 654, 350, str1, len1);
strcpy(str1,"(Sec.)");
len1 = strlen(str1);
XDrawString(dpy_popup_CTR, win, gc, 650, 370, str1, len1);
                                /* Change the foregroung color to draw the history
                                graph. */
XSetForeground(dpy_popup_CTR,gc,fill_pixel);
for(i=0;i<=20;i++)              /* Initialize the zero detact array. */
        zero_i[i]=-1;
for(i=0;i<=20;i++)
{                               /* Load the local array with the selected system
                                counter's values. */
        if(Pos==1)
                Local[i]=vc[C_line].VFirst[i];
        else if(Pos==2)
                Local[i]=vc[C_line].V2nd[i];
        else if(Pos==3)
                Local[i]=vc[C_line].V3rd[i];
}
        for(i=0;i<=20;i++)      /* Find maximum value. */
        {
                if(temp<Local[i])
                        temp=Local[i];
                if(Local[i]==0)
                        zero_i[i]=i;
        }
        high_val=temp;
        Low_val=Local[0];
        for(i=0;i<=20;i++)      /* Find minimum value. */
        {
                if(Low_val > Local[i])
                        Low_val=Local[i];
        }
        for(i=0;i<=20;i++)
        {                       /* Adjust the counter's values to display on the
                                drawing area. */
                y[i]=temp-Local[i];
                if(y[i] > base)
                        flag_base=1;
        }
        if(flag_base==1)        /* If counter value is out of range of the drawing
                                area then adjust again. */
        {
                ok=0;
                flag_base=0;
                for(i=0;i<=20;i++)
                {
                        if(y[i]<=999)   /* If counter value is under 999. */
                        { ok=1;div=10;}
                        else
                        {ok=0;break;}
                }
                if(ok==0)
                {
                        for(i=0;i<=20;i++)
                        if(y[i]<=9999) /* If counter value is under 9999. */
                        { ok=1;div=100;}
                        else
```

```
                {ok=0;break;}
        }
        if(ok==0)
        {
                for(i=0;i<=20;i++)
                if(y[i]<=99999) /* If counter value is under 99999. */
                { ok=1;div=10000;}
                else
                {ok=0;break;}
        }
        if(ok==0)
        {
                for(i=0;i<=20;i++)
                if(y[i]<=999999) /* If counter value is under 999999. */
                { ok=1;div=100000;}
                else
                {ok=0;break;}
        }
        for(i=0;i<=20;i++)
        res[i]=( (float)y[i]/(float)div );
        multiply=1.0;
        for(i=0;i<=20;i++)
        {
                if(res[i]<=0.9)
                multiply=10.0;
                else
                break;
        }
        for(i=0;i<=20;i++)
        {
                res[i]=res[i]*multiply;
                y[i]=(int)res[i];
        }

}                       /*end flag_base. */

add=0;
ok=0;
                        /* If the values of counter is very low then adjust
                        the scale. */
for(i=0;i<=20;i++)
{
        if(y[i]<=9)     /* If counter value is under 9. */
        {ok=1;add=343;}
        else
        {ok=0;break;}
}
if(ok==0)
{
        for(i=0;i<=20;i++)
        {
        if(y[i]<=19)    /* If counter value is under 19. */
        {ok=1;add=335;}
        else
        {ok=0;break;}
        }
}
if(ok==0)
{
        for(i=0;i<=20;i++)
        {
        if(y[i]<=39)    /* If counter value is under 39. */
        {ok=1;add=315;}
        else
        {ok=0;break;}
        }
}
if(ok==0)
{
        for(i=0;i<=20;i++)
        {
        if(y[i]<=60)    /* If counter value is under 60. */
        {ok=1;add=294;}
        else
        {ok=0;break;}
        }
}
if(ok==0)
{
        for(i=0;i<=20;i++)
```

```
                {
                if(y[i]<=70)    /* If counter value is under 70. */
                {ok=1;add=284;}
                else
                {ok=0;break;}
                }
        }
        if(ok==0)
        {
                for(i=0;i<=20;i++)
                {
                if(y[i]<=80)    /* If counter value is under 80. */
                {ok=1;add=274;}
                else
                {ok=0;break;}
                }
        }
        if(ok==0)
        {
                for(i=0;i<=20;i++)
                {
                if(y[i]<=90)    /* If counter value is under 90. */
                {ok=1;add=264;}
                else
                {ok=0;break;}
                }
        }
        if(ok==0)
        {
                for(i=0;i<=20;i++)
                {
                if(y[i]<=100)   /* If counter value is under 100. */
                {ok=1;add=254;}
                else
                {ok=0;break;}
                }
        }
        if(ok==0)
        {
                for(i=0;i<=20;i++)
                {
                if(y[i]<=110)   /* If counter value is under 110. */
                {ok=1;add=244;}
                else
                {ok=0;break;}
                }
        }
        if(ok==0)
        {
                for(i=0;i<=20;i++)
                {
                if(y[i]<=120)   /* If counter value is under 120. */
                {ok=1;add=234;}
                else
                {ok=0;break;}
                }
        }
        if(ok==0)
        {
                for(i=0;i<=20;i++)
                {
                if(y[i]<=130)   /* If counter value is under 130. */
                {ok=1;add=224;}
                else
                {ok=0;break;}
                }
        }
        if(ok==1)
                for(i=0;i<=20;i++)
                y[i]=y[i]+add; /* Add the calculated scale factor to the counter
                                values to scale the graph values. */
j=1;
for(i=0;i<=9;i++)
{
        if( (y[j] >=170) && (y[j] <= 200) )
                y[j]=y[j]-100;
        else if( (y[j] >= 201) && (y[j] <= 359) )
                y[j]=y[j]-100;
        j=j+2;
}
```

```
for(i=0;i<=20;i++)              /* Adjust the zero value of counter according to the
                                drawing area. */
{
        if(zero_i[i] != 1)
                y[i]=350;
}
strcpy(str1,"");
x=7;
for(i=0;i<=20;i++)
{                               /* Print the vertices for the graph. */

        XDrawString(dpy_popup_CTR, win, gc, x, y[i]+6, str1,strlen(str1));
        x=x+30;
}
x=10;
for(i=0;i<=19;i++)
{                               /* Connect all the vertices by using lines. */
        XDrawLine(dpy_popup_CTR,win,gc,x,y[i],x+30,y[i+1]);
        x=x+30;
}
                                /* Reset color to black. */
XSetForeground(dpy_popup_CTR,gc,fill_pixel2);
                                /* Print Maximum & Minimum values. */
 strcpy(str3,"Max ");
strcat(str3,lab_name);
 strcpy(str4,"Min ");
strcat(str4,lab_name);
 sprintf(str5,"%d",high_val);
 strcat(str3,str5);
 XDrawString(dpy_popup_CTR, win, gc, 545, 18, str3,strlen(str3));
 sprintf(str5,"%d",Low_val);
 strcat(str4,str5);
 XDrawString(dpy_popup_CTR, win, gc, 545, 38, str4,strlen(str4));
                                /* Print sampling time. */
tim_v=time(0);
 today_ctr=localtime(&tim_v);
 asctime (Sc, "%S", today_ctr);
 asctime (Mn, "%M", today_ctr);
 asctime (Hr, "%H", today_ctr);
 strcpy(intvl1,Hr);
 strcat(intvl1,":");
 strcat(intvl1,Mn);
 strcat(intvl1,":");
 strcat(intvl1,Sc);
 sec=atoi(Sc);
 MIN=atoi(Mn);
 if(sec >= 20)                  /* Calculate the the seconds. */
 {
         sec=sec-20;
         strcpy(intvl2,Hr);
         strcat(intvl2,":");
         sprintf(Mn,"%d",MIN);
         strcat(intvl2,Mn);
         strcat(intvl2,":");
         sprintf(Sc,"%d",sec);
         strcat(intvl2,Sc);

         strcat(total_intvl,intvl2);
         strcat(total_intvl," - ");
         strcat(total_intvl,intvl1);
                                /* Print time interval. */
        XDrawString(dpy_popup_CTR, win, gc,110,370,total_intvl,
                                  strlen(total_intvl));
 }
 else if(MIN >= 1)              /* Calculate the the minutes. */
 {
         ANS=20-sec;
         S_MIN=MIN;
         MIN=(MIN*60)-ANS;
         if(MIN >= 60)
         {
                 while(1)
                 {
                         MIN=MIN-60;
                         if(MIN < 60)
                                 break;
                 }
         }
         sec=MIN;
         --S_MIN;
```

```
                strcpy(intvl2,Hr);
                strcat(intvl2,":");
                sprintf(Mn,"%d",S_MIN);
                strcat(intvl2,Mn);
                strcat(intvl2,":");
                sprintf(Sc,"%d",sec);
                strcat(intvl2,Sc);
                strcat(total_intvl,intvl2);
                strcat(total_intvl,"-");
                strcat(total_intvl,intvl1);
                                        /* Print time interval. */
                XDrawString(dpy_popup_CTR, win, gc,110,370,total_intvl,
                                          strlen(total_intvl));
        }
                                        /* Reset the cursor back to normal. */
attrs.cursor=None;
XChangeWindowAttributes(dpy_wch,XtWindow(toplevel_ctr),CWCursor,&attrs);
}                                       /* end show_utilization2. */
                                        /* This routine initialize the graph data structures.
                                        */
void initialize_graph()
{
int i,j;
        for(i=0;i<=25;i++)
        {                               /* Initialize data structure for previous counter's
                                        values. */
                prm[i].first_pval=0;
                prm[i].v2nd_pval=0;
                prm[i].v3rd_pval=0;
        }

        for(i=0;i<=23;i++)
        {
                for(j=0;j<=22;j++)
                {                       /* Initialize data structure for current counter's
                                        values. */
                        vc[i].VFirst[j]=0;
                        vc[i].V2nd[j]=0;
                        vc[i].V3rd[j]=0;
                }
        }
        for(i=0;i<=3;i++)
        {
                Counter[i].Line=0;
                Counter[i].Val_pos=0;
        }
        for(i=0;i<=24;i++)
        {                               /* Initialize data structure for counter's values.
                                        */
                Prev[i].PFirst=0;
                Prev[i].P2nd=0;
                Prev[i].P3rd=0;
        }
                                        /* Initialize index variables for the graph data
                                        structures. */
idn0=0;idn1=0;idn2=0;idn3=0;idn4=0;idn5=0;idn6=0;idn7=0;idn8=0;idn9=0;idn10=0;idn11=0;idn1
2=0;idn13=0;idn14=0;idn15=0;idn16=0;idn17=0;idn18=0;idn19=0;idn20=0;idn21=0;idn22=0;idn23=
0;
}


#include "button_callbacks"             /*  Call back routines for system counters push
                                        buttons. */
                                        /* This routine executes as child process. Its jobs
                                        is to capture the curses's screen for the system
                                        counter data and send them to parent process through
                                        pipe. */
void child1()
{
int n,y,g;
char item[MAX];
FILE *fp;
int GetData=0;

initialization();                       /* Initialization for last word structure. */

                                        /* Wait for system counter windows set up. */
read(pipe_getdata[0],&GetData,sizeof(GetData));
while(GetData != 4);
```

```
                                        /* Execute the monitor command and get its output
                                        through pipe. */
if( (fp=popen("monitor -f < /z/rsyedra/motif/part1/f","r"))==NULL)
{printf("popen error"); exit(0);}

while((fgets(item,MAX,fp)) !=NULL)
{
        n=strlen(item);
        for(y=0;y<=n;y++)
        {
                g=(int)item[y];         /* Get the intger value of characters. */
                if(g==85)               /* ASCII value of character 'T'. */
                {
                        f_item(fp,item,y,n);
                }
        }
}
pclose(fp);
exit(0);
}                                       /* end child1. */
                                        /* This routine extracts each character from the
                                        output line. */
void f_item(FILE *fp,char *item,int y,int n)
{
        if(y<(n-1))
                fn2(y,item);
        while(fgets(item,MAX,fp)!=NULL)
        {
                n=strlen(item);
                if(n==1)                /* If this line is blank. */
                        ++line_ctr;
                else
                {
                        if(first_time)  /* if this routine execute first time.*/
                                fn2(0,item);
                        else
                                fn3(0,item);
                }
        }
}
                                        /* This routine use screen control characters to
                                        tracks the cursor movement and extract the data from
                                        curses screen. */
void fn2(int i,char *item)
{
int val,g,ap,data[7];
int position,d=0;
int ntab=0;

if(line_ctr==0)                         /* If screen line = 0 : line = globel. */
{
        while(item[i]!='m')
                ++i;
        cur_pos_ctr=cur_pos_ctr+15;     /*Position the cursor for Total User Time. */
        i=i+2;                          /* skip e. */
        if(item[i]==27)                 /* Control character ^[. */
        {
                i=i+2;                  /* Skip control character [. */
                val=get_values(item,&i);    /* Get values for cursor. */
                g=(int)item[i];         /* Get Control character. */
                set_cursor(g,-1,val);   /* -1 = no tab */
                ++i;
                while((item[i]!=27)&&(item[i]!=9)) /* Control character ^[ && tab. */
                {
                        put_line((int)item[i]); /* Store data in the line structure. */

                        ++i;
                }
        }
        else if(item[i]==9)             /*If control character is tab. */
        {
                while(item[i]==9)       /* Control character tab. */
                {
                        set_cursor(NULL,1,NULL);/* Tab is ON. */
                        ++i;
                }
                ntab=0;
                while((item[i]!=27)&&(item[i]!=9))/* Control character ^[ && tab. */
                {
                        put_line((int)item[i]); /* Store data in the line structure. */
```

```
                ++i;
        }
}
                                /* Capture Dirty page values. */
while(item[i] !='s')
        ++i;
cur_pos_ctr=cur_pos_ctr+18;     /*Position the cursor over the required data. */
++i;
if(item[i]==27)                 /* Control character is ^[. */
{
        i=i+2;                  /* Skip Control character [. */
        val=get_values(item,&i);    /* Get values for cursor. */
        g=(int)item[i];         /* Get Control character. */
        set_cursor(g,-1,val);   /* -1 = no tab. */
        ++i;
        while((item[i]!=27)&&(item[i]!=9))  /* Control character ^[ && tab. */
        {
                put_line((int)item[i]); /* Store data in the line structure. */

                ++i;
        }
}
else if(item[i]==9)             /* Control character is tab. */
{
        while(item[i]==9)       /* Ignore tab. */
                ++i;
        while(item[i] != 27)    /* Control character ^[. */
                data[d++]=item[i++];
        position=49;            /*Position the cursor at column 49. */
        while(--d >= 0)
        {                       /* Store data in the line structure. */
                acct[line_ctr].sl[position]=data[d];
                --position;
        }
}
                                /* Capture Semaphore ops values. */
while(item[i] !='s')
        ++i;
cur_pos_ctr=cur_pos_ctr+16;     /*Position the cursor. */
++i;
if(item[i]==27)                 /* Control character is ^[. */
{
        while(item[i] !=C)      /* Ignore control character C. */
                ++i;
        ++i;                    /* skip C. */
        while(item[i] != 13)    /* Control character is ^M. */
                data[d++]=item[i++];
        position=75;            /*Position the cursor at column 75. */
        while(--d >= 0)
        {                       /* Store data in the line structure. */
                acct[line_ctr].sl[position]=data[d];
                --position;
        }
}
else if(item[i]==9)             /* Ignore control character tab. */
{
        while(item[i]==9)       /* Control character tab. */
                ++i;
        while(item[i] != 13)    /* Control character ^M. */
                data[d++]=item[i++];
        position=75;            /*Position the cursor at column 75. */
        while(--d >= 0)
        {                       /* Store data in the line structure. */
                acct[line_ctr].sl[position]=data[d];
                --position;
        }
}
send_data();                    /* Send stsyem counter data to parent process. */
--line_ctr;                     /* Increment the line. */
cur_pos_ctr=1;                  /* Reset the cursor position. */
return;
}                               /* end if line = 0. */
                                /* Capture screen lines 1 to 22. */
if( (line_ctr > 0) && (line_ctr <23) )
{                               /* Get the index value for last char of the
                                parameter. */
        ap=wrd[1].ary[line_ctr];
        while(item[i]!= wrd[ap].lastch[1])
                ++i;
        ++ap;                   /* Increment for 2nd parm. */
```

```
i=i+wrd[1].skp[line_ctr];  /* Cursor skip values after last character. */
                           /* Cursor position value. */
cur_pos_ctr=cur_pos_ctr+wrd[1].cur_add[line_ctr];

if(item[i]==27)            /* Control character is ^[. */
{
        i=i+2;             /* Skip the control character [. */
        val=get_values(item,&i);/* Get values for cursor. */
        g=(int)item[i];/* Get control character. */
        set_cursor(g,-1,val);/* -1 = no tab */
        ++i;
                           /* Control characters ^[ && tab. */
        while((item[i]!=27)&&(item[i]!=9))
        {                  /* Store data in the line structure. */
        put_line((int)item[i]);
                ++i;
        }
}
else if(item[i]==9)        /* Control character is tab. */
{                          /* Ignore control character tab. */
        while(item[i]==9)
                ++i;
                           /* Control character is ^[. */
        while(item[i] != 27)
                data[d++]=item[i++];
        position=23;       /*Position the cursor at column 23. */
        while(--d >= 0)
        {                  /* Store data in the line structure. */
                acct[line_ctr].sl[position]=data[d];
                --position;
        }
        cur_pos_ctr=24;/*Position the cursor. */
}
                           /* Capture 2nd part. */
while(item[i]!= wrd[ap].lastch[1])
        ++i;
i=i+wrd[2].skp[line_ctr];/* Cursor skip values after last character.*/
                           /* Cursor position value. */
cur_pos_ctr=cur_pos_ctr+wrd[2].cur_add[line_ctr];

if(item[i]==9)             /* Ignore the control character tab. */
{                          /* Control character is tab. */
        while(item[i]==9)
                ++i;
                           /* Control character ^M. */
        while(item[i] != 13)
                data[d++]=item[i++];
        if(d==6)
                           /*Position the cursor at column 50. */
                position=50;
        else               /*Position the cursor at column 49. */
                position=49;
        while(--d >= 0)
        {                  /* Store data in the line structure. */
                acct[line_ctr].sl[position]=data[d];
                --position;
        }
}
else if(item[i]==27)       /* Control character is ^[. */
{                          /* Ignore control character C. */
        while(item[i] !=C)
                ++i;
        ++i;               /* Skip control character C. */
                           /* Control character ^M. */
        while(item[i] != 13)
                data[d++]=item[i++];
        if(d==6)           /*Position the cursor at column 50. */
                position=50;
        else               /*Position the cursor at column 49. */
                position=49;
        while(--d >= 0)
        {
                acct[line_ctr].sl[position]=data[d];
                --position;
        }
}
send_data();               /* Send stsyem counter data to parent process. */
++line_ctr;                /* Increment the line. */
cur_pos_ctr=1;             /* Reset the cursor position. */
return;
```

```
}                               /* block r1 ends. */
                                /* Capture Free Page Recs  : last Line. */
if(line_ctr==23)                /* If screen line = 23.*/
{
        ap=wrd[1].ary[line_ctr];
        while(item[i]!= wrd[ap].lastch[1])
                ++i;
        ++ap;
        i=i+wrd[1].skp[line_ctr];
        cur_pos_ctr=cur_pos_ctr+wrd[1].cur_add[line_ctr];
        if(item[i]==27)         /* Control character is ^[. */
        {
                i=i+2;          /* Skip the control character [. */
                                /* Get values for cursor. */
                val=get_values(item,&i);
                g=(int)item[i];/* Get control character. */
                                /* -1 = no tab. */
                set_cursor(g,-1,val);
                ++i;
                                /* Control character ^[ && tab. */
                while((item[i]!=27)&&(item[i]!=9))
                {               /* Store data in the line structure. */
                        put_line((int)item[i]);
                        ++i;
                }
        }
        else if(item[i]==9)     /* Control character is tab. */
        {                       /* Ignore tab. */
                while(item[i]==9)
                        ++i;
                                /* Control character is ^[. */
                while(item[i] != 27)
                        data[d++]=item[i++];
                position=23;
                while(--d >= 0)
                {               /* Store data in the line structure. */
                        acct[line_ctr].sl[position]=data[d];
                        --position;
                }
                cur_pos_ctr=24; /*Position the cursor. */
        }

                                /* Capture Message Ops values. */
        while(item[i]!= wrd[ap].lastch[1])
                ++i;
                                /* Cursor skip values after last character.*/
        i=i+wrd[2].skp[line_ctr];
                                /* Cursor position. */
        cur_pos_ctr=cur_pos_ctr+wrd[2].cur_add[line_ctr];
        if(item[i]==9)          /* Ignore the tab. */
        {                       /* Control character is tab. */
                while(item[i]==9)
                        ++i;
                                /* Control character is ^[. */
                while(item[i] != 27)
                        data[d++]=item[i++];
                position=49;
                while(--d >= 0)
                {
                        acct[line_ctr].sl[position]=data[d];
                        --position;
                }
                cur_pos_ctr=50; /*Position the cursor. */
        }
        else if(item[i]==27)    /* Control character is ^[. */
        {                       /* Ignore control character C. */
                while(item[i] !=C)
                        ++i;
                ++i;            /* Skip C. */
                                /* Control character is ^[. */
                while(item[i] != 27)
                        data[d++]=item[i++];
                position=49;
                while(--d >= 0)
                {
                        acct[line_ctr].sl[position]=data[d];
                        --position;
                }
        }
        send_data();            /* Send stsyem counter data to parent process. */
```

```
                                /* Now Go to first line (line=0) and get data. */
                if(item[i] == 27)       /* Control character is ^[. */
                {
                        ++i;
                        if(item[i]==91)/* Control character is [. */
                        {
                                ++i;
                                        /* Control character is H. */
                                if(item[i]==72)
                                {       /* Go to top of the screen. */
                                line_ctr=0;
                                ++i;
                                /* Reset cursor. */
                                cur_pos_ctr=1;
                                }
                        }
                        i=i+2;
                        val=get_values(item,&i);
                        g=(int)item[i];
                        set_cursor(g,-1,val);
                        i=i+3;
                        val=get_values(item,&i);
                        g=(int)item[i];
                        set_cursor(g,-1,val);
                        ++i;
                                /* ^[ && tab */
                        while((item[i] !=27)&&(item[i]!=9)&& (item[i]!=10))
                        {
                                put_line((int)item[i]);
                                ++i;
                        }
                        if(item[i]==9) /* Control character is tab. */
                        {
                                ntab=d=0;
                                        /* Control character is tab. */
                                while(item[i++]==9)
                                        ++ntab;
                                if(ntab==4)
                                {
                                    while((item[i]!=10)&&(item[i]!=27)&&(item[i]!=9))
                                                data[d++]=item[i++];
                                    position=49;
                                    while(--d >= 0)
                                    {
                                            acct[line_ctr].sl[position]=data[d];
                                            --position;
                                    }
                                }
                                ntab=d=0;
                                        /*Position the cursor. */
                                cur_pos_ctr=50;
                        }
                        else if(item[i]==27)
                        {
                                i=i+2;
                                val=get_values(item,&i);
                                g=(int)item[i];
                                set_cursor(g,-1,val);
                                ++i;
                        }
                                        /* Get data for Semaphore Ops. */
                        while((item[i]!=10)&&(item[i]!=27)&&(item[i]!=9))
                        {
                                put_line((int)item[i]);
                                ++i;
                        }
                }

                send_data();            /* Send stsyem counter data to parent process. */
                ++line_ctr;             /* Increment the line. */
        }                               /* block 1 ends */
        first_time=0;                   /* Reset flag to zero, indicating that now it is not
                                        first time. */
}                                       /* end fn2. */
                                        /* This routine executes only when the curses
                                        updates the screen. First it get the control
                                        character and cursor value and then jump on that
                                        screen position to get the required data. */
void fn3(int i,char *item)
```

```
{
int data[4];                            /* Screen data array. */
int g,d,val=0;
int ntab=0;
int position=0;
int save_i,flag_data=1;
while(TRUE)
{
        if(item[i]==27)                 /* Control character is '['.  */
        {
                i=i+2;
                val=get_values(item,&i);
                g=(int)item[i];         /* Get control character. */
                if((g==B)||(g==A))
                send_data();            /* Send stsyem counter data to parent process. */
                set_cursor(g,-1,val);   /* Set cursor according to control character. */
                ++i;
                while(1)
                {                       /* Read the data until end of screen line. */
                        while((item[i]!=10)&&(item[i]!=27)&&(item[i]!=8) )
                        {
                                flag_data=1;
                                /* If control character is tab. */
                                if(item[i]==9)
                                {
                                        save_i=i;
                                        while(item[i]==9)
                                        {
                                                ++ntab;
                                                ++i;
                                        }
                                        if(ntab==4)
                                        {/* Position the cursor. */
                                                cur_pos_ctr = 49;
                                                flag_data=0;
                                                ntab=0;
                                        }
                                        else
                                        {
                                                i=save_i;
                                                flag_data=1;
                                        }
                                }
                                if(flag_data)
                                {
                                        if(item[i]==9)
                                                ++cur_pos_ctr;
                                        else
                                                put_line((int)item[i]);
                                        ++i;
                                }
                                flag_data=1;
                        }
                        if(item[i]==27)
                        {
                                i=i+2;
                                /* If cursor is at last line. */
                                if(item[i]==H)
                                {       /* Send stsyem counter data to parent process. */
                                        send_data();
                                        ++l;
                                        /*Reset the screen line. */
                                        line_ctr=0;
                                        /* Reset the cursor position. */
                                        cur_pos_ctr=1;
                                        /* exit from while B. */
                                        break;
                                }
                                val=get_values(item,&i);
                                        /* Get control character. */
                                g=(int)item[i];
                          if((g==B)||(g==A))
                                        /* Send stsyem counter data to parent process. */
                                        send_data();
                                        /* Set cursor according to control character. */
                                set_cursor(g,-1,val);
                                ++i;
                        }
                        else if(item[i]==10)
                                        /* Send stsyem counter data to parent process. */
```

```
                        send_data();
                                /* Increment the line. */
                        ++line_ctr;
                        return;
                }
                else if(item[i]==8)
                {               /* If control character is ^H. */
                        while(item[i]==8)
                        {       /* Decrement the cursor. */
                                --cur_pos_ctr;
                                ++i;
                        }
                }
        }
}
else if(item[i]==8)             /* ^H .*/
{
        while(1)
        {                       /*Control character is ^H. */
                while(item[i]==8)
                {
                        --cur_pos_ctr;
                        ++i;
                }
                while((item[i]!=10)&&(item[i]!=27))
                {
                        if(item[i]==9)
                        {
                                save_i=i;
                                while(item[i]==9)
                                {
                                        ++ntab;
                                        ++i;
                                }
                                if(ntab==4)
                                {
                                        cur_pos_ctr = 49;
                                        flag_data=0;
                                        ntab=0;
                                }
                                else
                                {
                                        i=save_i;
                                        flag_data=1;
                                }
                        }
                        if(flag_data)
                        {
                                if(item[i]==9)
                                        ++cur_pos_ctr;
                                else
                                        put_line((int)item[i]);
                                ++i;
                        }
                        flag_data=1;
                }
                if(item[i]==27)
                {
                        i=i+2;
                        if(item[i]==H)
                        {
                                send_data();
                                ++i;
                                line_ctr=0;
                                cur_pos_ctr=1;
                                /* exit from while(1). */
                                break;
                        }
                        val=get_values(item,&i);
                        g=(int)item[i];
                        if((g==B)||(g==A))
                                send_data();
                        set_cursor(g,-1,val);
                        ++i;
                }
                else if(item[i]==10)
                {
                        send_data();
                        ++line_ctr;
                        return;
```

```
                }
            }
        }
        else if( (item[i]==32)||(item[i]==9)||
        ( (item[i]>=48) && (item[i] <= 57) )  )
        {
            while(TRUE)
            {
                d=ntab=0;
                while(item[i]==9)
                {
                    ++ntab;
                    ++i;
                }
                if(ntab==4)
                {
                    while( (item[i] != 10) &&(item[i] != 27) )
                        data[d++]=item[i++];
                    if(d==6)
                    {       position=50;
                            cur_pos_ctr=51;}
                    else
                    {       position=49;
                            cur_pos_ctr=50;}
                    while(--d >= 0)
                    {
                        acct[line_ctr].sl[position]=data[d];
                        --position;
                    }
                    d=ntab=0;
                }               /* end if */
                else if( (item[i]==32)||
                ( (item[i]>=48) && (item[i] <= 57) )  )
                {
                    while( (item[i]= 32)||
                    ( (item[i]>=48) && (item[i] <= 57) )  )
                    {
                        put_line((int)item[i]);
                        ++i;
                    }
                }
                if( (item[i]==27)||(item[i]==8) )
                    break;
                if(item[i]==10)
                {
                    send_data();
                    ++line_ctr;
                    return;
                }
            }
        }
        else if(item[i]==10)
        {
            ++line_ctr;
            return;
        }
        }
}                                   /* end fn3 */
                                    /* This routine gives cursor position values that
                                    are associated with the control characters. */
int get_values(char *item,int *i)
{
int d[2];
int g=0;
d[0]=d[1]=-1;
        g=(int)item[*i];
        if((g!=C)&&(g!=D)&&(g!=A)&&(g!=B))
        {
            d[0]=g;
            ++*i;
            g=(int)item[*i];
            if((g!=C)&&(g!=D)&&(g!=A)&&(g!=B))
            {
                d[1]=g;
                ++*i;
            }
        }
        return(convt_ctr(d[0],d[1]));
}
```

```
                                        /* This routine store actual data of system counter
                                        into the screen line structure so that actual data
                                        can be extracted. */
void put_line(int data)
{                                       /* Put data on line and increment the cursor. */
        acct[line_ctr].sl[cur_pos_ctr++]=data;
}
                                        /* This routine set the directions of cursor. These
                                        directions are forward, backward, downward, and
                                        upward. */
void set_cursor(int ct_char,int tab,int val)
{
        if(tab == -1)                   /* Only control characters. */
        {
                if(ct_char == C)        /* Go forward. */
                        cur_pos_ctr=cur_pos_ctr+val;
                else if(ct_char == D)
                                        /* Go backward. */
                        cur_pos_ctr=cur_pos_ctr-val;
                else if(ct_char == B)
                                        /* Go downward. */
                        line_ctr=line_ctr+val;
                else if(ct_char == A)
                                        /* Go upward. */
                        line_ctr=line_ctr-val;
        }
}
                                        /* This routine converts ASCII values from curses
                                        screen to integer values. */
int convt_ctr(int j,int k)
{
int T[2];
int i,cn;

        T[0]=j;
        T[1]=k;
        if(T[0]==-1) return(1);
        for(i=0;i<2;i++)
        {
                switch(T[i])
                {
                        case 48:
                        cn=0;
                        break;

                        case 49:
                        cn=1;
                        break;

                        case 50:
                        cn=2;
                        break;
                        case 51:
                        cn=3;
                        break;

                        case 52:
                        cn=4;
                        break;

                        case 53:
                        cn=5;
                        break;

                        case 54:
                        cn=6;
                        break;

                        case 55:
                        cn=7;
                        break;

                        case 56:
                        cn=8;
                        break;
                        case 57:
                        cn=9;
                        break;

                        default:
```

```
                        cn=-1;
                        break;

                }                       /* end switch */
                T[i]=cn;
        }                               /* end for */

        if(T[1]!=-1)
                T[0]=( T[0]*10 )+T[1]; /* Calculate the values. */
        return(T[0]);
}
                                        /* This routine initialize the array and data
                                        structures utilized in this child process. */
void initialization()
{
int j,k,p;
                                        /* String contains the last or 2nd last alphabet of
                                        the system counter on the monitor screen. */
char alphabet[50]="*isenssstssinssstttylosmsssssstsHsseseBnBtdsdsp";

k=1;

                                        /* Initialize values for index in lastch. */
for(j=1;j<=46;j+=2)
        wrd[1].ary[k++]=j;
                                        /* Initialize last char for the parameters. */
for(j=1;j<=46;j++)
        wrd[j].lastch[1]=alphabet[j];

                                        /* Initialize skip values for the cursor so that
                                        cursor can be position on the data. */
for(j=1;j<=23;j++)
{
        wrd[1].skp[j]=1;
        wrd[2].skp[j]=1;
}
wrd[1].skp[1]=3;wrd[1].skp[6]=2;wrd[1].skp[11]=3;wrd[2].skp[11]=3;
wrd[2].skp[16]=3;wrd[1].skp[18]=3;wrd[1].skp[19]=2;wrd[2].skp[23]=2;
                                        /* Cursor addition values to use in skip values. */
wrd[1].cur_add[1]=17;
wrd[2].cur_add[1]=11;wrd[1].cur_add[2]=10;wrd[2].cur_add[2]=17;
wrd[1].cur_add[3]=15;wrd[2].cur_add[3]=12;wrd[1].cur_add[4]=13;
wrd[2].cur_add[4]=18;wrd[1].cur_add[5]=14;wrd[2].cur_add[5]=11;
wrd[1].cur_add[6]=9;wrd[2].cur_add[6]=19;wrd[1].cur_add[7]=14;
wrd[2].cur_add[7]=12;wrd[1].cur_add[8]=13;wrd[2].cur_add[8]=20;
wrd[1].cur_add[9]=16;wrd[2].cur_add[9]=10;wrd[1].cur_add[10]=17;
wrd[2].cur_add[10]=14;wrd[1].cur_add[11]=12;wrd[2].cur_add[11]=15;
wrd[1].cur_add[12]=16;wrd[2].cur_add[12]=20;wrd[1].cur_add[13]=10;
wrd[2].cur_add[13]=15;wrd[1].cur_add[14]=5;wrd[2].cur_add[14]=16;
wrd[1].cur_add[15]=5;wrd[2].cur_add[15]=14;wrd[1].cur_add[16]=6;
wrd[2].cur_add[16]=15;wrd[1].cur_add[17]=5;wrd[2].cur_add[17]=12;
wrd[1].cur_add[18]=14;wrd[2].cur_add[18]=13;wrd[1].cur_add[19]=11;
wrd[2].cur_add[19]=14;wrd[1].cur_add[20]=12;wrd[2].cur_add[20]=15;
wrd[1].cur_add[21]=13;wrd[2].cur_add[21]=13;wrd[1].cur_add[22]=11;
wrd[2].cur_add[22]=21;wrd[1].cur_add[23]=14;wrd[2].cur_add[23]=14;
                                        /* Initialize screen line structure. */
for(p=0;p<=23;p++)
{
acct[p].sl[19]=0;
acct[p].sl[20]=0;
acct[p].sl[21]=0;
acct[p].sl[22]=0;
acct[p].sl[23]=0;
acct[p].sl[45]=0;
acct[p].sl[46]=0;
acct[p].sl[47]=0;
acct[p].sl[48]=0;
acct[p].sl[49]=0;
acct[p].sl[50]=0;
acct[p].sl[73]=0;
acct[p].sl[74]=0;
acct[p].sl[75]=0;
}

}
                                        /* This routine send the system counter data to the
                                        parent process through pipe. */
void send_data()
{
int scr_line;
```

```
long val1,val2,val3;
int line_zero=0;
scr_line=line_ctr;                      /* Get the screen line number. */

if(scr_line==0)                         /* If line number is zero. */
{                                       /* Convert the values of screen line structure into
                                        integer values. */
        val1=convert_data(0,0,acct[0].sl[20],acct[0].sl[21],acct[0].sl[22],
        acct[0].sl[23]);
        val2=convert_data(0,0,0,acct[0].sl[47],acct[0].sl[48],acct[0].sl[49]);
        val3=convert_data(0,0,0,acct[0].sl[73],acct[0].sl[74],acct[0].sl[75]);
                                        /* Send the line number. */
        write(pipe_cntrB[1],&line_zero,sizeof(line_zero));
                                        /* Send the first value. */
        write(pipe_cntrB[1],&val1,sizeof(val1));
                                        /* Send the 2nd value. */
        write(pipe_cntrB[1],&val2,sizeof(val2));
                                        /* Send the 3rd value. */
        write(pipe_cntrB[1],&val3,sizeof(val3));
}
else                                    /* If line number is not zero the send only first
                                        and 2 nd values. */
{
        val1=convert_data(0,acct[scr_line].sl[19],acct[scr_line].sl[20],
        acct[scr_line].sl[21],acct[scr_line].sl[22],acct[scr_line].sl[23]);

        if( (acct[scr_line].sl[50] >= 48)&&(acct[scr_line].sl[50] <=57) )
        {                               /* If the values are in six digits. */
                val2=convert_data(acct[scr_line].sl[45],acct[scr_line].sl[46],
                acct[scr_line].sl[47],acct[scr_line].sl[48],acct[scr_line].sl[49],
                acct[scr_line].sl[50]);
        }
        else
        {
                val2=convert_data(0,acct[scr_line].sl[45],
                acct[scr_line].sl[46],acct[scr_line].sl[47],
                acct[scr_line].sl[48],acct[scr_line].sl[49]);
        }
                write(pipe_cntrB[1],&scr_line,sizeof(scr_line));
                write(pipe_cntrB[1],&val1,sizeof(val1));
                write(pipe_cntrB[1],&val2,sizeof(val2));
}
}
                                        /* This routine converts the values of screen line
                                        data structure into integer values. It takes the
                                        position of digits and then multiply by its weights.
                                        These weights are 10, 100, 1000, 10000, and 100000.
                                        */
int convert_data(int a,int b,int c,int d,int e,int f)
{
int T[7];
int i,cn,p;
T[0]=T[1]=T[2]=T[3]=T[4]=T[5]=T[6]=T[7]=0;
T[0]=a;T[1]=b;T[2]=c;T[3]=d;T[4]=e;T[5]=f;
p=5;/* If values are in five digits. */
if(a)
        p=6;
for(i=0;i<=p;i++)
{
        switch(T[i])
        {
                case 48:                /* If ASCII value is 0. */
                cn=0;
                break;

                case 49:                /* If ASCII value is 1. */
                cn=1;
                break;

                case 50:                /* If ASCII value is 2. */
                cn=2;
                break;

                case 51:                /* If ASCII value is 3. */
                cn=3;
                break;

                case 52:                /* If ASCII value is 4. */
                cn=4;
                break;
```

```
                case 53:                /* If ASCII value is 5. */
                cn=5;
                break;

                case 54:                /* If ASCII value is 6. */
                cn=6;
                break;

                case 55:                /* If ASCII value is 7. */
                cn=7;
                break;

                case 56:                /* If ASCII value is 8. */
                cn=8;
                break;

                case 57:                /* If ASCII value is 9. */
                cn=9;
                break;

                default:
                cn=0;
                break;

        }                               /* end switch. */
        T[i]=cn;
}                                       /* end for. */
                                        /* Multiply values by its weights. */
T[0]=(T[0]*100000)+(T[1]*10000)+(T[2]*1000)+(T[3]*100)+(T[4]*10)+T[5];
return(T[0]);
}
  /*****************************************************************/
 /*                          File: storeit                        */
/*****************************************************************/

                                        /* Declare the index variables for storing the
                                        system and user time. */
int cind0,cind1,cind2,cind3,cind4,cind5,cind6,cind7,cind8,cind9;
int cind10,cind11,cind12,cind13,cind14,cind15,cind16,cind17,cind18,cind19;
int cind20,cind21,cind22,cind23;

                                        /* This routine store the values of the system time
                                        and user time in a data structure so that they can
                                        be used to draw the graphs whenever user presses the
                                        history button on the CPU window. This routine
                                        sample only 20 values. */
void cpu_value_store(int line,int V1st,int V2nd)
{
int k;

        switch(line)                    /* Determine the screen line number. */
        {
                case 0:                 /* If line is 0. */
                                        /* Store system time. */
                Vcpu[0].sys_time[cind0]=V1st;
                                        /* Store user time. */
                Vcpu[0].usr_time[cind0++]=V2nd;
                if(cind0 > 20)
                {
                        for(k=0;k<=19;k++)
                        {               /* Adjust the data structure. */
                                Vcpu[0].sys_time[k]=Vcpu[0].sys_time[k+1];
                                Vcpu[0].usr_time[k]=Vcpu[0].usr_time[k+1];
                        }
                        cind0=20;       /* Take only 20 values. */
                }
                break;

                case 1:                 /* If line is 1. */
                Vcpu[1].sys_time[cind1]=V1st; /* Store system time. */
                Vcpu[1].usr_time[cind1++]=V2nd; /* Store user time. */
                if(cind1 > 20)
                {
                        for(k=0;k<=19;k++)
                        {               /* Adjust the data structure. */
                                Vcpu[1].sys_time[k]=Vcpu[1].sys_time[k+1];
                                Vcpu[1].usr_time[k]=Vcpu[1].usr_time[k+1];
                        }
                        cind1=20;       /* Take only 20 values. */
```

```
}
break;

case 2:                 /* If line is 2. */
Vcpu[2].sys_time[cind2]=Vlst;
Vcpu[2].usr_time[cind2++]=V2nd;
if(cind2 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[2].sys_time[k]=Vcpu[2].sys_time[k+1];
                Vcpu[2].usr_time[k]=Vcpu[2].usr_time[k+1];
        }
        cind2=20;
}
break;

case 3:                 /* If line is 3. */
Vcpu[3].sys_time[cind3]=Vlst;
Vcpu[3].usr_time[cind3++]=V2nd;
if(cind3 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[3].sys_time[k]=Vcpu[3].sys_time[k+1];
                Vcpu[3].usr_time[k]=Vcpu[3].usr_time[k+1];
        }
        cind3=20;
}
break;

case 4:                 /* If line is 4. */
Vcpu[4].sys_time[cind4]=Vlst;
Vcpu[4].usr_time[cind4++]=V2nd;
if(cind4 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[4].sys_time[k]=Vcpu[4].sys_time[k+1];
                Vcpu[4].usr_time[k]=Vcpu[4].usr_time[k+1];
        }
        cind4=20;
}
break;

case 5:                 /* If line is 5. */
Vcpu[5].sys_time[cind5]=Vlst;
Vcpu[5].usr_time[cind5++]=V2nd;
if(cind5 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[5].sys_time[k]=Vcpu[5].sys_time[k+1];
                Vcpu[5].usr_time[k]=Vcpu[5].usr_time[k+1];
        }
        cind5=20;
}
break;

case 6:                 /* If line is 6. */
Vcpu[6].sys_time[cind6]=Vlst;
Vcpu[6].usr_time[cind6++]=V2nd;
if(cind6 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[6].sys_time[k]=Vcpu[6].sys_time[k+1];
                Vcpu[6].usr_time[k]=Vcpu[6].usr_time[k+1];
        }
        cind6=20;
}
break;
case 7:                 /* If line is 7. */
Vcpu[7].sys_time[cind7]=Vlst;
Vcpu[7].usr_time[cind7++]=V2nd;
if(cind7 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[7].sys_time[k]=Vcpu[7].sys_time[k+1];
```

```
                Vcpu[7].usr_time[k]=Vcpu[7].usr_time[k+1];
        }
        cind7=20;
}
break;
case 8:                 /* If line is 8. */
Vcpu[8].sys_time[cind8]=V1st;
Vcpu[8].usr_time[cind8++]=V2nd;
if(cind8 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[8].sys_time[k]=Vcpu[8].sys_time[k+1];
                Vcpu[8].usr_time[k]=Vcpu[8].usr_time[k+1];
        }
        cind8=20;
}
break;
case 9:                 /* If line is 9. */
Vcpu[9].sys_time[cind9]=V1st;
Vcpu[9].usr_time[cind9++]=V2nd;
if(cind9 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[9].sys_time[k]=Vcpu[9].sys_time[k+1];
                Vcpu[9].usr_time[k]=Vcpu[9].usr_time[k+1];
        }
        cind9=20;
}
break;
case 10:                /* If line is 10. */
Vcpu[10].sys_time[cind10]=V1st;
Vcpu[10].usr_time[cind10++]=V2nd;
if(cind10 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[10].sys_time[k]=Vcpu[10].sys_time[k+1];
                Vcpu[10].usr_time[k]=Vcpu[10].usr_time[k+1];
        }
        cind10=20;
}
break;

case 11:                /* If line is 11. */
Vcpu[11].sys_time[cind11]=V1st;
Vcpu[11].usr_time[cind11++]=V2nd;
if(cind11 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[11].sys_time[k]=Vcpu[11].sys_time[k+1];
                Vcpu[11].usr_time[k]=Vcpu[11].usr_time[k+1];
        }
        cind11=20;
}
break;
case 12:                /* If line is 12. */
Vcpu[12].sys_time[cind12]=V1st;
Vcpu[12].usr_time[cind12++]=V2nd;
if(cind12 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[12].sys_time[k]=Vcpu[12].sys_time[k+1];
                Vcpu[12].usr_time[k]=Vcpu[12].usr_time[k+1];
        }
        cind12=20;
}
break;
case 13:                /* If line is 13. */
Vcpu[13].sys_time[cind13]=V1st;
Vcpu[13].usr_time[cind13++]=V2nd;
if(cind13 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[13].sys_time[k]=Vcpu[13].sys_time[k+1];
                Vcpu[13].usr_time[k]=Vcpu[13].usr_time[k+1];
```

```
        }
        cind13=20;
}
break;
case 14:                /* If line is 14. */
Vcpu[14].sys_time[cind14]=V1st;
Vcpu[14].usr_time[cind14++]=V2nd;
if(cind14 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[14].sys_time[k]=Vcpu[14].sys_time[k+1];
                Vcpu[14].usr_time[k]=Vcpu[14].usr_time[k+1];
        }
        cind14=20;
}
break;
case 15:                /* If line is 15. */
Vcpu[15].sys_time[cind15]=V1st;
Vcpu[15].usr_time[cind15++]=V2nd;
if(cind15 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[15].sys_time[k]=Vcpu[15].sys_time[k+1];
                Vcpu[15].usr_time[k]=Vcpu[15].usr_time[k+1];
        }
        cind15=20;
}
break;
case 16:                /* If line is 16. */
Vcpu[16].sys_time[cind16]=V1st;
Vcpu[16].usr_time[cind16++]=V2nd;
if(cind16 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[16].sys_time[k]=Vcpu[16].sys_time[k+1];
                Vcpu[16].usr_time[k]=Vcpu[16].usr_time[k+1];
        }
        cind16=20;
}
break;
case 17:                /* If line is 17. */
Vcpu[17].sys_time[cind17]=V1st;
Vcpu[17].usr_time[cind17++]=V2nd;
if(cind17 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[17].sys_time[k]=Vcpu[17].sys_time[k+1];
                Vcpu[17].usr_time[k]=Vcpu[17].usr_time[k+1];
        }
        cind17=20;
}
break;
case 18:                /* If line is 18. */
Vcpu[18].sys_time[cind18]=V1st;
Vcpu[18].usr_time[cind18++]=V2nd;
if(cind18 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[18].sys_time[k]=Vcpu[18].sys_time[k+1];
                Vcpu[18].usr_time[k]=Vcpu[18].usr_time[k+1];
        }
        cind18=20;
}
break;
case 19:                /* If line is 19. */
Vcpu[19].sys_time[cind19]=V1st;
Vcpu[19].usr_time[cind19++]=V2nd;
if(cind19 > 20)
{
        for(k=0;k<=19;k++)
        {
                Vcpu[19].sys_time[k]=Vcpu[19].sys_time[k+1];
                Vcpu[19].usr_time[k]=Vcpu[19].usr_time[k+1];
        }
        cind19=20;
```

```
                .
                break;
                case 20:                /* If line is 20. */
                Vcpu[20].sys_time[cind20]=V1st;
                Vcpu[20].usr_time[cind20++]=V2nd;
                if(cind20 > 20)
                {
                        for(k=0;k<=19;k++)
                        {
                                Vcpu[20].sys_time[k]=Vcpu[20].sys_time[k+1];
                                Vcpu[20].usr_time[k]=Vcpu[20].usr_time[k+1];
                        }
                        cind20=20;
                }
                break;
                case 21:                /* If line is 21. */
                Vcpu[21].sys_time[cind21]=V1st;
                Vcpu[21].usr_time[cind21++]=V2nd;
                if(cind21 > 20)
                {
                        for(k=0;k<=19;k++)
                        {
                                Vcpu[21].sys_time[k]=Vcpu[21].sys_time[k+1];
                                Vcpu[21].usr_time[k]=Vcpu[21].usr_time[k+1];
                        }
                        cind21=20;
                }
                break;
                case 22:                /* If line is 22. */
                Vcpu[22].sys_time[cind22]=V1st;
                Vcpu[22].usr_time[cind22++]=V2nd;
                if(cind22 > 20)
                {
                        for(k=0;k<=19;k++)
                        {
                                Vcpu[22].sys_time[k]=Vcpu[22].sys_time[k+1];
                                Vcpu[22].usr_time[k]=Vcpu[22].usr_time[k+1];
                        }
                        cind22=20;
                }
                break;
                case 23:                /* If line is 23. */
                Vcpu[23].sys_time[cind23]=V1st;
                Vcpu[23].usr_time[cind23++]=V2nd;
                if(cind23 > 20)
                {
                        for(k=0;k<=19;k++)
                        {
                                Vcpu[23].sys_time[k]=Vcpu[23].sys_time[k+1];
                                Vcpu[23].usr_time[k]=Vcpu[23].usr_time[k+1];
                        }
                        cind23=20;
                }
                break;

        }                       /* switch end */
}
  /*****************************************************************/
 /*                        File: cpu_window                       */
/*****************************************************************/
Display *dpy_cpu_popup;         /* Display ID of pop up window. */
Colormap cmap;                  /* Color information. */
                                /* This routine executes as the child process of the
                                main program. The purpose of this routine is to
                                create and display CPU window, capture the CPU
                                activity, and send the data through the pipe. */
void chld2(argc, argv)
int argc;
char *argv[];
{
int cpu_go=0;
                                /* String used for frame names. */
String name_fr[] = { "fr1",  "fr2", "fr3", "fr4", "fr5",  fr6",  "fr7", "fr8","fr9",
"fr10","fr11", "fr12","fr13", "fr14","fr15" "fr16",
"fr17" "fr18","fr19" "fr20","fr21" "fr22","fr23","fr24"};
                                /* String used for scale widgets names. */
String name_sc[] = {
"sc1",  "sc2", "sc3", "sc4", "sc5", "sc6",  "sc7", "sc8", "sc9", "sc10","sc11", "sc12",
"sc13","sc14","sc15", "sc16","sc17", "sc18","sc19","sc20","sc21", "sc22","sc23","sc24",
"sc25",  "sc26", "sc27", "sc28", "sc29", "sc30",  "sc31", "sc32","sc33", "sc34","sc35",
```

```
"sc36", "sc37", "sc38","sc39","sc40","sc41", "sc42","sc43","sc44",
"sc45","sc46","sc47","sc48");

                                        /* Labels for each CPU. */
String name_cpu[] = {
"\n CPU #0\n\n\n\n  S   U","\n CPU #1\n\n\n\n  S   U",
"\n CPU #2\n\n\n\n  S   U","\n CPU #3\n\n\n\n  S   U",
"\n CPU #4\n\n\n\n  S   U","\n CPU #5\n\n\n\n  S   U",
"\n CPU #6\n\n\n\n  S   U","\n CPU #7\n\n\n\n  S   U",
"\n CPU #8\n\n\n\n  S   U","\n CPU #9\n\n\n\n  S   U",
"\n CPU #10\n\n\n\n  S   U","\n CPU #11\n\n\n\n  S   U",
"\n CPU #12\n\n\n\n  S   U","\n CPU #13\n\n\n\n  S   U",
"\n CPU #14\n\n\n\n  S   U","\n CPU #15\n\n\n\n  S   U",
"\n CPU #16\n\n\n\n  S   U","\n CPU #17\n\n\n\n  S   U",
"\n CPU #18\n\n\n\n  S   U","\n CPU #19\n\n\n\n  S   U",
"\n CPU #20\n\n\n\n  S   U","\n CPU #21\n\n\n\n  S   U",
"\n CPU #22\n\n\n\n  S   U","\n CPU #23\n\n\n\n  S   U");
                                        /* Create title string for CPU window. */
XmString Title_st2=XmStringCreateSimple("CPU Activities for each Processor");
                                        /* Bitmap graphics for CPU window. */
char str_ledgend[50]="/z/rsyedna/motif/part1/cpu_ledgend.icon";
Pixmap pixmap_ledg;                     /* Bitmap information. */
Widget  form_a[25];
Widget  form,form2,frame,frame2,frame3;
int k1,k2,Ar,i,j,n=0;
Arg args[2];
int sc_num,ic;
Widget dialog;
Widget Hbttn[25];                       /* Buttons for History. */
XWindowAttributes xwa;
int go2=10;
Cursor cursor;
XSetWindowAttributes attrs;
Widget toplevel3,form_main,save_btn;
int n1,n2,n3;
int sln,ksys,kus;
                                        /* Create pipes for use in the child process. */
if(pipe(p)<0)
        printf("error in creating pipe. \n");
if(pipe(pipe2)<0)
        printf("error in creating pipe. \n");

                                        /* Create child process to capture the CPU activity.
                                        */
if( (pid=fork()) == -1)
        {printf("can't fork a child \n"); exit(0);}
if(pid==0)
{                                       /* For capture monitor screen 1 */
        child_cpu_act();                /* Get the CPU data. */
}                                       /* fork end */
else
{                                       /* Parent process. */
                                        /* Create the root window. */
toplevel3 = XtVaAppInitialize (&app, "XMemo", NULL, 0,&argc, argv, NULL, NULL);
                                        /* Get display ID of the root window. */
dpy = XtDisplay (toplevel3);

                                        /* Create top level form and attach it to the root
                                        window. */
form=XtVaCreateWidget("main_window",
xmFormWidgetClass,  toplevel3,          /* Parent */
XmNfractionBase, 10,
XmNwidth, 970, XmNheight, 650,          /* Size of form. */
NULL);
                                        /* Get the background and foreground color
                                        information. */
XtVaGetValues(form, XmNforeground, &fg, XmNbackground, &bg, NULL);
                                        /* Create bitmap graphics for the CPU window. */
pixmap = XmGetPixmap (XtScreen (form), cur_bitmap, fg, bg);
pixmap2   XmGetPixmap (XtScreen (form), cur_bitmap2, fg, bg);
pixmap_ledg = XmGetPixmap (XtScreen (form), str_ledgend, fg, bg);

                                        /* Create label for icon and attach it to the form.
                                        */
XtVaCreateManagedWidget (cur_bitmap, xmLabelWidgetClass, form,
XmNlabelType, XmPIXMAP,
XmNlabelPixmap, pixmap,                 /* Shape of icon. */
                                        /* Attach position of this widget with its parent
                                        widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
```

```
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 1,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 1, NULL);

                                        /* Create the title label for CPU window. *
XtVaCreateManagedWidget("cpu",
xmLabelWidgetClass,form,
XmNlabelString, Title_st2,
                                        /* Attach position of this widget with its parent
                                        widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 1,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 5,
XmNalignment,XmALIGNMENT_BEGINNING,     /* Left justification. */
NULL);
                                        /* Create label to define system and user time. */
XtVaCreateManagedWidget("cpu",
xmLabelWidgetClass,form,
XmNlabelType, XmPIXMAP,
XmNlabelPixmap, pixmap_ledg,
                                        /* Attach position of this widget with its parent
                                        widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 1,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 5,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 7, NULL);
XtVaCreateManagedWidget("cpu",
xmLabelWidgetClass,form,
XmNlabelString, Title_st4,
                                        /* Attach position of this widget with its parent
                                        widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 1,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 7,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 9, NULL);
                                        /* Create CPU utilization push button. */
save_btn=XtVaCreateManagedWidget ("button",
xmPushButtonWidgetClass, form,          /* Parent. */
XmNlabelType, XmPIXMAP,
XmNlabelPixmap, pixmap2,                /* Button graphic. */
                                        /* Attach position of this widget with its parent
                                        widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 1,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 9,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 10, NULL);
                                        /* Set the call back routine for CPU utilization
                                        push button. */
XtAddCallback(save_btn, XmNactivateCallback, bcall, NULL);
                                        /* Create frame widget. */
frame = XtVaCreateWidget("frame",
xmFrameWidgetClass, form,               /* Parent. */
XmNshadowType, XmSHADOW_ETCHED_OUT,
                                        /* Attach position of this widget with its parent
                                        widget. */
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 1,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 10,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 10, NULL);

                                        /* Create form widget as a child of frame widget. */
form2 = XtVaCreateWidget("form2",
xmFormWidgetClass,  frame,
XmNfractionBase, 12,
XmNwidth, 200, XmNheight, 200, NULL);
k1=0;
k2=6;
Ar=0;
sc_num=0;
for(j=0;j<=23;j++)
{
        if(j>=12)
        { k1=6; k2=12;Ar=12;}
                                        /* Create frames for CPU scale widgets. */
        frame_a[j] = XtVaCreateWidget(name_fr[j],
        xmFrameWidgetClass, form2,
        XmNshadowType, XmSHADOW_IN,
                                        /* Attach position of this widget with its parent
                                        widget. */
```

```
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, k1,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, k2,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, (j-Ar),
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, (j-Ar)+1, NULL);
                                /* Create from widget. */
form_a[j] = XtVaCreateWidget("form_a",
xmFormWidgetClass,  frame_a[j],
XmNfractionBase, 3, NULL);
                                /* Create label widgets for name of CPUs. */
XtVaCreateManagedWidget(name_cpu[j],
xmLabelWidgetClass,form_a[j],
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 1,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 1, NULL);
                                /* Create history buttons for CPU window. */
Hbttn[j]=XtVaCreateManagedWidget ("H",
xmPushButtonWidgetClass, form_a[j],
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 1,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 2,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 3, NULL);
                                /* Set the background color of history buttons to
                                green. */
XtVaSetValues(Hbttn[j],XtVaTypedArg,XmNbackground, XmRString,"green",6,NULL);
                                /* Set the user data of history buttons. */
XtVaSetValues(Hbttn[j],XmNuserData,j,NULL);
                                /* Set the call back routine for history buttons. */
XtAddCallback(Hbttn[j], XmNactivateCallback, show_cpu_stat, NULL);
for(ic=0;ic<=1;ic++)
{                               /* Create the scale widgets and attach them to form
                                widgets. */
        stline[j].scale_a[sc_num] =
        XtVaCreateManagedWidget ("scale",
        xmScaleWidgetClass, form_a[j], /* Parent widgets. */
                                /* Title string for scale. */
        XtVaTypedArg, XmNtitleString, XmRString, " ", 3,
        XmNmaximum,         100, /* Maximum value of scale widget. */
        XmNminimum,         0,   /* Minimum value of scale widget. */
        XmNvalue,           0,   /* Default value. */
        XmNshowValue,       True,/* Display value on the scale widget. */
        XmNscaleWidth, 13,       /* width of scale widget. */
        XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 1,
        XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 3,
        XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, ic,
        XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, ic+1,
        NULL);
        ++sc_num;               /* Increment the scale widget number. */
}
sc_num=0;
                                /* Create tick marks on the scale widgets. */
for (i = 0; i < MAX_VAL_MARK; i++)
XtVaCreateManagedWidget ("-", xmLabelGadgetClass, stline[j].scale_a[1], NULL);

XtManageChild(form_a[j]);
XtManageChild(frame_a[j]);
}
                                /* Manage all form widgets. */
XtManageChild(form2);
XtManageChild(frame);
XtManageChild(form);

flag3=0;
                                /* Realize the root window so that it could appear
                                on the screen. */
XtRealizeWidget (toplevel3);

while(1)
{                               /* Wait for user to press the CPU Activities push
                                button. */
        read(cpu_pipe[0],&cpu_go,sizeof(cpu_go));
        if(cpu_go==20)
                break;
}
                                /* signal the child process to start capturing the
                                CPU data. */
write(pipe2[1],&go2,sizeof(go2));

for(;;){
```

```
                if(flag3==0){           /* If cursor is in normal shape then change the
                                        shape of cursor to busy state. */
                flag3=1;
                cursor=XCreateFontCursor(dpy,XC_watch);
                attrs.cursor=cursor;
                XChangeWindowAttributes(dpy,XtWindow(toplevel3),CWCursor,&attrs);
                }                       /* end flag. */
                                        /* Update the CPU window. */
                XmUpdateDisplay(toplevel3);
                                        /* Select the interested X events and dispatch these
                                        events to X server. */
                if(XCheckMaskEvent(dpy,ButtonPressMask | ButtonReleaseMask ,
                ExposureMask,&event))
                {       XtDispatchEvent(&event); }
                                        /* Read the line number. */
                read(p[0],&sln,sizeof(sln));
                                        /* Read system time values. */
                read(p[0],&ksys,sizeof(ksys));
                                        /* Read user time values. */
                read(p[0],&kus,sizeof(kus));
                                        /* Store these values for display the history
                                        graphics. */
                cpu_value_store(sln,ksys,kus);
                                        /* Display the system time on the scale widgets. */
                XmScaleSetValue(stline[sln].scale_a[0],ksys);
                                        /* Display the user time on the scale widgets. */
                XmScaleSetValue(stline[sln].scale_a[1],kus);
                                        /* Update the CPU utilization data structure. */
                ++cpu_utilization[sln].cpu_util;
                if(flag3==1){           /* If the cursor is in busy state then change the
                                        cursor back to its normal shape. */
                        flag3=2;
                        attrs.cursor=None;
                        XChangeWindowAttributes(dpy,XtWindow(toplevel3),CWCursor,
                        &attrs);
                        }               /* end flag. */
                }                       /* end for. */
        }                               /* Parent end. */
}                                       /* child2 end. */
                                        /* This routine executes as a call back for history
                                        buttons. It creates pop up window for history
                                        graphics. */
void
show_cpu_stat(w, client_data, call_data)
Widget w;
XtPointer client_data;                  /* Client call data. */
XtPointer call_data;                    /* Routine call data. */
{
XGCValues gcv;
GC gc;
Widget  form;
Arg args[2];
char temp_str[]="History for CPU #"; /* Title string for history window. */
XmString label_str;
Screen *scr_popup;
int cpu_number=0;
Widget dialog,draw,frame_pop1,frame_pop2,btn_down;
char buf[9]="History";
char cno[9];
                                        /* Get the CPU number from the history button. */
XtVaGetValues(w,XmNuserData,&cpu_number,NULL);
                                        /* Create title string for history window. */
sprintf(cno,"%d",cpu_number);
                                        /* Create pop up dialog shell. */
dialog = XtVaCreatePopupShell ("popup",
xmDialogShellWidgetClass, GetTopShell (w), /* Parent widget. */
XmNtitle,               buf,            /* Title string. */
XmNallowShellResize, False,
XmNdeleteResponse,   XmDESTROY,
XmNx, 200, XmNy, 290,                   /* Size of shell. */
NULL);
                                        /* Create the form widget. */
form=XtVaCreateWidget("main_window",
xmFormWidgetClass,  dialog,
XmNfractionBase, 40,
XmNwidth, 750, XmNheight, 400, NULL);
                                        /* Create the frame widget. */
frame_pop1 = XtVaCreateManagedWidget("frame",
xmFrameWidgetClass, form,
XmNshadowType, XmSHADOW_ETCHED_OUT,
```

```
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 2,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 40, NULL);
strcat(temp_str,cno);
label_str=XmStringCreateSimple(temp_str);
                                        /* Create label widget. */
XtVaCreateManagedWidget("lb",
xmLabelWidgetClass, frame_pop1,
XmNlabelString, label_str,
NULL);
XmStringFree(label_str);                /* Free up the string which is no longer in use. */
                                        /* Create frame widget. */
frame_pop2 = XtVaCreateManagedWidget("frame",
xmFrameWidgetClass, form,
XmNshadowType, XmSHADOW_ETCHED_OUT,
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 2,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 40,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 38, NULL);
                                        /* Create OK push button and attach it to the form.
                                        */
bttn_down=XtVaCreateManagedWidget ("OK",
xmPushButtonWidgetClass, form,
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 37,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 40,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 38,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 40, NULL);
                                        /* Set the user data of OK button. */
XtVaSetValues(bttn_down, XmNuserData, dialog, NULL);
                                        /* Set the call back routine. */
XtAddCallback(bttn_down,XmNactivateCallback, cpu_stat_popdown,NULL);
                                        /* Set the backgroung color of OK button to green.
                                        */
XtVaSetValues(bttn_down,XtVaTypedArg,XmNbackground,XmRString,"Green",7,NULL);
                                        /* Create the drawing area for history graphics. */
draw = XtVaCreateManagedWidget("draw",
xmDrawingAreaWidgetClass, frame_pop2,
XmNwidth, 750,
XmNheight, 400,
NULL);
dpy_cpu_popup = XtDisplay(draw);        /* Get the display ID of drawing area. */
                                        /* Get the color information of drawing area. */
cmap=DefaultColormapOfScreen(XtScreen(draw));
scr_popup = XtScreen(draw);             /* Get the screen ID of drawing area. */
                                        /* Set the drawing area as the work area of form
                                        widget. */
XtVaSetValues(form, XmNworkWindow, draw, NULL);
                                        /* Set the call back routine for drawing area. */
XtAddCallback(draw, XmNexposeCallback, show_cpu_graph, cpu_number);
                                        /* Create the graphics context and attach it to the
                                        user data of drawing area. */
gcv.foreground = BlackPixelOfScreen(scr_popup);
gc = XCreateGC(dpy_cpu_popup,
RootWindowOfScreen(scr_popup), GCForeground, &gcv);
XtVaSetValues(draw,XmNuserData,gc,NULL);
                                        /* Manage the form widget. */
XtManageChild(form);
                                        /* Pop up the history window on the screen. */
XtPopup (dialog, XtGrabNone);
}


                                        /* This routine draws the graphics on the CPU
                                        history window. */
void
show_cpu_graph(Widget w, XtPointer data,
        XmDrawingAreaCallbackStruct *cbk)
{
char  str1[75],str2[75];
GC gc;
Window win = XtWindow(w);
int len1,x,i,j;
int height,base=350;                    /* Base coordinates of history window. */
int C_line,Pos,Div,Point[22];
int y[22],local[22],temp;
int do_again,flag_base;
int add,div,ok=0;
float res,2*,multiply;
char str3[20],str4[20],str5[20];
```

```
XColor xcolour,spare;                   /* X Window color information. */
long int fill_pixel=1;                  /* Foreground color for the window. */
long int fill_pixel2=1;
long int fill_pixel3=1;
int zero i[23];
int SYS_val[23],USR_val[23];
int max_s=0;
int max_u=0;
int mlpy=0;
int cpu_num=(int)data;                  /* CPU number. */
flag_base=0;
                                        /* Clear the window. */
XClearWindow(dpy_cpu_popup,win);
                                        /* Allocate red color to the window for system time.
                                        */
XAllocNamedColor(dpy_cpu_popup,cmap,"Red",&xcolour,&spare);

fill_pixel=xcolour.pixel;               /* Save this color. */

                                        /* Allocate blue color to the window for user time.
                                        */
XAllocNamedColor(dpy_cpu_popup,cmap,"Blue",&xcolour,&spare);
fill_pixel2=xcolour.pixel;              /* Save this color. */

                                        /* Allocate black color to the window for axis
                                        lines. */
XAllocNamedColor(dpy_cpu_popup,cmap,"Black",&xcolour,&spare);

fill_pixel3=xcolour.pixel;              /* Save this color. */
                                        /* Get the graphics context values. */
XtVaGetValues(w, XmNuserData, &gc, NULL);

                                        /* Draw x and y axis on the window. */
XDrawLine(dpy_cpu_popup,win,gc,10,10,10,350);

XDrawLine(dpy_cpu_popup,win,gc,10,350,650,350);
strcpy(str1,"^");
len1 = strlen(str1);
XDrawString(dpy_cpu_popup, win, gc, 6, 20, str1, len1);
strcpy(str1,"% of Time");
len1 = strlen(str1);
                                        /* Draw the label of y-axis. */
XDrawString(dpy_cpu_popup, win, gc, 18, 19, str1, len1);
strcpy(str1,">");
len1 = strlen(str1);
XDrawString(dpy_cpu_popup, win, gc, 646, 357, str1, len1);
strcpy(str1,"Time");
len1 = strlen(str1);
                                        /* Draw the label of x-axis. */
XDrawString(dpy_cpu_popup, win, gc, 654, 350, str1, len1);
strcpy(str1,"(Sec.)");
len1 = strlen(str1);
XDrawString(dpy_cpu_popup, win, gc, 650, 370, str1, len1);
for(i=0;i<=20;i++)                      /* Find maximum values of system and user time. */
{
        if(max_s<Vcpu[cpu_num].sys_time[i])
                max_s=Vcpu[cpu_num].sys_time[i];
        if(max_u<Vcpu[cpu_num].usr_time[i])
                max_u=Vcpu[cpu_num].usr_time[i];
}

if(max_s < max_u)                       /* Find which one is larger. */
        max_s max_u;
                                        /* If maximum value is too low then adjust the
                                        graph's scale to draw the lines  */
switch(max_s)
{
        case 10:                        /* If maximum value is 10 then multiply by 34. */
        mlpy=34;
        break;

        case 20:                        /* If maximum value is 20 then multiply by 17. */
        mlpy=17;
        break;

        case 30:                        /* If maximum value is 30 then multiply by 11. */
        mlpy=11;
        break;

        case 40:                        /* If maximum value is 40 then multiply by 8. */
```

```
        mlpy=8;
        break;

        case 50:                        /* If maximum value is 50 then multiply by 5. */
        mlpy=5;
        break;

        case 60:                        /* If maximum value is 60 then multiply by 5. */
        mlpy=5;
        break;

        case 70:                        /* If maximum value is 70 then multiply by 5. */
        mlpy=5;
        break;

        case 80:                        /* If maximum value is 80 then multiply by 4. */
        mlpy=4;
        break;
        case 90:                        /* If maximum value is 90 then multiply by 3. */
        mlpy=3;
        break;

        case 100:                       /* If maximum value is 100 then multiply by 3. */
        mlpy=3;
        break;

        default:                        /* Default value is 34. */
        mlpy=34;
        break;
}
for(i=0;i<=20;i++)                      /* Get values for system & user time. */
{
        SYS_val[i]=base-(Vcpu[cpu_num].sys_time[i]*mlpy);
        USR_val[i]=base-(Vcpu[cpu_num].usr_time[i]*mlpy);
}

strcpy(str1,"");
len1 = strlen(str1);

                                        /* Set the color for system time. */
XSetForeground(dpy_cpu_popup,gc,fill_pixel);

x=10;
for(i=0;i<=20;i++)                      /* For system time. */
{                                       /* Draw the vertices on the graph. */
        XDrawString(dpy_cpu_popup, win, gc,x,SYS_val[i]+5, str1, len1);
        x=x+30;
}

x=10;
for(i=0;i<=19;i++)
{                                       /* Connect all vertices on the graph. */
        XDrawLine(dpy_cpu_popup,win,gc,x,SYS_val[i],x+30,SYS_val[i+1]);
        x=x+30;
}
                                        /* Line for system time. */
XDrawLine(dpy_cpu_popup,win,gc,560,15,580,15);
strcpy(str1,"System Time");             /* Print label for system time. */
len1 = strlen(str1);
XDrawString(dpy_cpu_popup, win, gc,583,20, str1, len1);
XSetForeground(dpy_cpu_popup,gc,fill_pixel2);
x=10;
strcpy(str1,"");
len1 = strlen(str1);
for(i=0;i<=20;i++)                      /* For user time. */
{                                       /* Draw the vertices on the graph. */
        XDrawString(dpy_cpu_popup, win, gc,x,USR_val[i]+5, str1, len1);
        x=x+30;
}
x=10;
for(i=0;i<=19;i++)
{                                       /* Connect all vertices on the graph. */
        XDrawLine(dpy_cpu_popup,win,gc,x,USR_val[i],x+30,USR_val[i+1]);
        x=x+30;
}
                                        /* Line for user time. */
XDrawLine(dpy_cpu_popup,win,gc,560,40,580,40);
strcpy(str1,"User Time");               /* Print label for user time. */
len1 = strlen(str1);
XDrawString(dpy_cpu_popup, win, gc,583,45, str1, len1);
```

```
                                        /* Reset the color to black */
XSetForeground(dpy_cpu_popup,gc,fill_pixel3);
}
                                        /* This is call back routine for the OK button on
                                        the history window. */
void
cpu_stat_popdown(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
Widget wid;
int cr_no=0;
XtVaGetValues(w, XmNuserData, &wid, NULL);
XtDestroyWidget(wid); /* Destroy this widget. */
}
                                        /* This routine executes as the child process. Its
                                        job is to capture the CPU data and send them to the
                                        parent process through pipe. */
void child_cpu_act(void)                /* For run monitor screen 1. */
{
int n,g,y;
int flag2=0;
int go=0;
char item[MAX];
FILE *fp;
                                        /* Wait for the user to press the CPU Activities
                                        push button on the main window. */
while(1)
{
        read(pipe2[0],&go,sizeof(go));
        if(go==10)
                break;
}
                                        /* Execute monitor command and take its output
                                        through pipe. */
if( (fp=popen("monitor -c1","r"))==NULL)
        {printf("popen error"); exit(0);}


while((fgets(item,MAX,fp)) !=NULL)
{
        n=strlen(item);                 /* Number of items. */
        for(y=0;y<=n;y++)
        {
                g=(int)item[y];         /* Convert into integer. */
                if(g==101)
                {
                        flag2=1;
                        cur_pos=78;     /* Default position of cursor. */
                        flag_ok=1;
                        cr(fp,item,y,n);/* Extract data from output. */
                        break;
                }
        }
        if(flag2)                       /* If done then exit. */
                break;
}
pclose(fp);                             /* Close the pipe. */
exit(0);
}
                                        /* This routine is the call back for CPU Report push
                                        button. This routine create the pop up window for
                                        CPU utilization graphics. */
void
bcall(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
XGCValues gcv;                          /* Graphics context value. */
GC gc;
Widget  form;
Arg args[2];
Colormap cmap_popup;                    /* Color information. */
Widget dialog,draw,frame_pop1,frame_pop2,bttn_down;
char buf[8]="Report";                   /* Title string. */
                                        /* Create the pop up shell. */
```

```
dialog = XtVaCreatePopupShell ("popup",
xmDialogShellWidgetClass, GetTopShell (w),
XmNtitle,            buf,              /* Title string. */
XmNallowShellResize, False,
XmNdeleteResponse, XmDESTROY,
XmNx,        250,                      /* Size of shell. */
XmNy,        290,
NULL);
                                       /* Create the form widget. */
form=XtVaCreateWidget("main_window",
xmFormWidgetClass,  dialog,
XmNfractionBase, 40,
XmNwidth, 750, XmNheight, 400, NULL);
                                       /* Create the frame widget. */
frame_pop1 = XtVaCreateManagedWidget("frame",
xmFrameWidgetClass, form,
XmNshadowType, XmSHADOW_ETCHED_OUT,
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 0,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 2,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 40, NULL);
                                       /* Create the title label for the window. */
XtVaCreateManagedWidget("CPU Utilization Report", xmLabelWidgetClass, frame_pop1, NULL);
                                       /* Create the frame widget. */
frame_pop2 = XtVaCreateManagedWidget("frame",
xmFrameWidgetClass, form,
XmNshadowType, XmSHADOW_ETCHED_OUT,
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 2,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 40,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 0,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 38, NULL);
                                       /* Create OK push button and attach it to the form.
                                       */
bttn_down=XtVaCreateManagedWidget ("OK",
xmPushButtonWidgetClass, form,
XmNtopAttachment, XmATTACH_POSITION, XmNtopPosition, 37,
XmNbottomAttachment, XmATTACH_POSITION, XmNbottomPosition, 40,
XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 38,
XmNrightAttachment, XmATTACH_POSITION, XmNrightPosition, 40, NULL);
                                       /* Set the background color of OK button to green.
                                       */
XtVaSetValues(bttn_down,XtVaTypedArg,XmNbackground,XmRString,"Green",7,NULL);
                                       /* Set the user data. */
XtVaSetValues(bttn_down, XmNuserData, dialog, NULL);
                                       /* Set the call back routine for OK button. */
XtAddCallback(bttn_down,XmNactivateCallback, Popdown,NULL);
                                       /* Create the drawing area widget. */
draw = XtVaCreateManagedWidget("draw",
xmDrawingAreaWidgetClass, frame_pop2,
XmNwidth, 750, XmNheight, 400, NULL);


dpy_popup = XtDisplay(draw);           /* Get the display ID of drawing area. */
scr_ptr_popup = XtScreen(draw);        /* Get the screen ID of drawing area. */
                                       /* Set the drawing area as a work area of form
                                       widget. */
XtVaSetValues(form, XmNworkWindow, draw, NULL);

                                       /* Create the graphics context of drawing area and
                                       attach it to the user data of the drawing area. */
gcv.foreground = BlackPixelOfScreen(scr_ptr_popup);
gc = XCreateGC(dpy_popup, RootWindowOfScreen(scr_ptr_popup), GCForeground, &gcv);
XtVaSetValues(draw,XmNuserData,gc,NULL);

                                       /* Get the color information. */
cmap_popup=DefaultColormapOfScreen(XtScreen(draw));

                                       /* Set the call back routine for drawing area
                                       widget. */
XtAddCallback(draw, XmNexposeCallback, show_utilization, cmap_popup);


                                       /* Manage the form. */
XtManageChild(form);
                                       /* Display the pop up window on the screen. */
XtPopup (dialog, XtGrabNone);
}
```

```
                                        /* This is the call back routine for OK button on
                                        the CPU utilization window. This routine close the
                                        CPU utilization window. */
void
Popdown(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
Widget wid;
int i;
int flag_util=0;
        for(i=0;i<=24;i++){             /* Clear the CPU utilization structure if it reaches
                                        to the edge of the graph. */
                if(cpu_utilization[i].cpu_util != 300 )
                        flag_util=1;}
        if( flag_util == 0 )            /*All reach to 300 values. */
        {       for(i=0;i<=24;i++)      /* Initialize the structure. */
                        cpu_utilization[i].cpu_util=0;
        }
        XtVaGetValues(w, XmNuserData, &wid, NULL);
        XtDestroyWidget(wid);           /* Destroy the utilization window. */

}
                                        /* This routine initialize the CPU utilization
                                        structure. */
void report_ini()
{
int i;
        for(i=0;i<=24;i++)
                cpu_utilization[i].cpu_util=0;
}
                                        /* This routine draws bar graphs on the CPU
                                        utilization window. */
void
show_utilization(Widget w, XtPointer data,
        XmDrawingAreaCallbackStruct *cbk)
{
char  str1[75],str2[75];
char no1[25],no2[25];
char cno1[75]="CPU #";
char cno2[75]="CPU #";
GC gc;
Window win = XtWindow(w);
int len1,x,y;
int i,x_dist,flag_low;
int height,base=350;
int high,low,get_cpu_low,get_cpu_high;
long int fill_pix=1;
long int fill_pix2=1;
XColor xcolr,spr;
Colormap cmap_popup=(Colormap)data;
                                        /* Allocate block color to window. */
XAllocNamedColor(dpy_popup,cmap_popup,"Black",&xcolr,&spr);
fill_pix=xcolr.pixel;                   /* Save this color. */
XAllocNamedColor(dpy_popup,cmap_popup,"Black",&xcolr,&spr);
fill_pix2=xcolr.pixel;                  /* Save this color. */
high=low=get_cpu_low=get_cpu_high=0;
                                        /* Get the user data. */
XtVaGetValues(w, XmNuserData, &gc, NULL);

                                        /* Draw x and y axis. */
XDrawLine(dpy_popup,win,gc,10,10,10,350);
XDrawLine(dpy_popup,win,gc,10,350,650,350);

strcpy(str1,"^");
len1 = strlen(str1);
XDrawString(dpy_popup, win, gc, 6, 20, str1, len1);
strcpy(str1,"Utilization");
len1 = strlen(str1);
XDrawString(dpy_popup, win, gc, 18, 19, str1, len1);
strcpy(str1,">");
len1 = strlen(str1);
x=30; y=50;
XDrawString(dpy_popup, win, gc, 646, 357, str1, len1);
strcpy(str1,"CPU#");
len1   strlen(str1);
XDrawString(dpy_popup, win, gc, 660, 378, str1, len1);
strcpy(str1,"0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 ");
strcpy (str2,"17 18 19 20 21 22 23");
```

```
strcat(str1,str2);
len1 = strlen(str1);
                                        /* Draw CPU number. */
XDrawString(dpy_popup, win, gc, 25, 369, str1, len1);

low=cpu_utilization[0].cpu_util;
                                        /* Set yellow color for the drawing area. */
XSetForeground(dpy_popup,gc,fill_pix);
x_dist=2;

                                        /* Calculate the x-axis distance. */
for(i=0;i<=23;i++)
{
        if( (i>=10)&&(i<=16) )
                x_dist+=29;
        else if( (i>=17)&&(i<=20) )
                x_dist+=31;
        else if( (i>=21)&&(i<=23) )
                x_dist+=29;
        else
                x_dist+=20;

        height=cpu_utilization[i].cpu_util;

        if(height>297)
                height=292;

        if(high<cpu_utilization[i].cpu_util)
        {
                high=cpu_utilization[i].cpu_util;
                get_cpu_high=i;
        }
        if(low>cpu_utilization[i].cpu_util)
        {
                low=cpu_utilization[i].cpu_util;
                get_cpu_low=i;
        }
        y=base-height;
                                        /* Draw bar graphs. */
        XFillRectangle(dpy_popup,win,gc,x_dist,y,6,height);
}
XSetForeground(dpy_popup,gc,fill_pix2);/* Reset color to black. */
sprintf(no1,"%d",get_cpu_high);
sprintf(no2,"%d",get_cpu_low);
strcat(cno1,no1);
strcat(cno2,no2);
strcat(cno1," is utilized heavily, ");
strcat(cno2," is utilized lightly");
strcat(cno1,cno2);
len1 = strlen(cno1);

                                        /* Print the utilization message. */
XDrawString(dpy_popup, win, gc, 140, 45, cno1, len1);
XDrawRectangle(dpy_popup,win,gc,135,19,563,36);
}


                                        /* This routine updates the screen line number and
                                        take monitor output to calculate the system and user
                                        time. */
void cr(FILE *fp,char *item,int y,int n)
{
int i;
        if(y<(n-1))
                fn(y,n,item);

        while((fgets(item,MAX,fp)) !=NULL)
        {
                n=strlen(item);
                if(n==1)
                {
                        --line;         /* Increment screen line number. */
                        flag=0;
                }
                else
                        fn(--line,n,item);/* Track the cursor movements. */
        }

}
```

```
                                        /* This routine tracks the movements of the cursor
                                        by using control characters and get the data for
                                        each CPU. */
void fn(int i,int n,char *item)
{
int g=0;
int a[2],k,tab=0;
int sys,user,val=0;
int control_char=0;
user=sys=0;
sp_ctr=0;
++i;
while(i<=(n-1))
{
        for(k=0;k<2;k++)
                a[k]=-1;
        g=value(i,n,item);
        if(g==-1)
                break;
        if(g==27)
        {
                if(dflag)
                {                       /* If control character is C. */
                        if(control_char==C)
                                        /* Increment the cursor. */
                                cur_pos=cur_pos+val;
                                        /* If control character is D. */
                        else if(control_char==D)
                                        /* Decrement the cursor. */
                                cur_pos=cur_pos-val;
                        dflag=0;
                        val=0;
                }

                ++i;
                g=value(i++,n,item);    /* Get the control character. */
                if(g==-1)
                        break;
                if(g==91)
                {
                        g=value(i++,n,item);
                        if(g==-1)
                                break;
                        if(g==72)
                        {       flag=1;
                                dflag=0;
                                flag_ok=0;
                                line=0;
                                cur_pos=1;/* Default cursor position. */
                        }
                        else
                        {
                                if((g!=C)&&(g!=D)&&(g!=A)&&(g!=B))
                                {       /* Get the cursor movement value. */
                                        /* Store 1st digit */
                                        a[0]=g;
                                        g=value(i++,n,item);
                                        if(g==-1)
                                                break;
                                        if((g!=C)&&(g!=D)&&(g!=A)&&(g!=B))
                                        {/* Store 2nd digit */
                                                a[1]=g;
                                                g=value(i++,n,item);
                                                if(g==-1)
                                                        break;
                                        }
                        }
                                        /* If cursor is moving in any direction. */
                        if(g==C||g==D||g==A||g==B)
                        {               /* Expact data */
                                dflag=1;
                                if(g==C)
                                        /* Control character is C. */
                                        control_char=C;

                                if(g==D)/* Control character is D. */
                                        control_char=D;
                                if(g==A)
                                        /* Control character is A. */
```

```
                                control_char=A;
                        if(g==B)
                                /* Control character is B. */
                                control_char=B;
                                /* Get the cursor movement values. */
                        val=convt(a[0],a[1]);
                                /* If control character is A then go up. */
                        if(control_char==A)
                        line=22-val; dflag=0; val=0;}
                                /* If control character is B then go down. */
                        if(control_char==B)
                                {line=line+val; dflag=0; val=0;}
                }
        }
    }
}
else if(g==10)
{
        if(dflag)
        {
                if(control_char==C)
                        cur_pos=cur_pos+val;
                else if(control_char==D)
                        cur_pos=cur_pos-val;
        }
        ++i;
        flg_space=0;
        if(!flag)
                ++line;
        return;
}
else if(g==61||g==45)                   /* If data is system or user time (= , -). */
{
        dflag=0;
        if(i==0)
                vflag=1;
        if(g==61)
                ++sys;                  /* Increment system time. */
        else
                ++user;                 /* Increment user time. */
        ++i;
        g=(int)item[i];
        if( (g!=61)&&(g!=45) )
        {
                ctr_char(control_char,sys.user,line,flag,val,g);
                flag=0;
                vflag=0;
                sys=0;
                user=0;
                control_char=0;
                val=0;
        }
}
else if(g==H_CONT)
{
        if(dflag)
        {
                if(control_char==C)
                        cur_pos=cur_pos+val;
                else if(control_char==D)
                        cur_pos=cur_pos-val;
        }
        flag=0;
        while( (g=(int)item[i])==8 )
        {                               /* If control character is return character. */
                --cur_pos;              /* Decrement cursor. */
                ++i;
        }
        control_char=H_CONT;
}
else if(g==32)
{                                       /* If control character is space. */
        vflag=1;
        dflag=0;
        sp_ctr=0;


        while( (g=(int)item[i])==32 )
        {
                ++sp_ctr;
```

```
                ++i;
        }

        ctr_char(control_char,sys,user,line,flag,val,g);
        flag=0;
        vflag=0;
        sys=0;
        user=0;
        control_char=0;
        val=0;

}
else if(g==9)
{                                       /* If control character is TAB. */
        dflag=0;
        vflag=1;
        if(flag_ok)                     /* If tab on okstate line. */
        {
                val=val+9;
                cur_pos=cur_pos-val;
                flag_ok=0;
        }

                                        /* Findout the cursor's current position. */
        if( (cur_pos>=1)&&(cur_pos<=7) )
                cur_pos=8;
        else if( (cur_pos>=8)&&(cur_pos<=15) )
                cur_pos=16;
        else if( (cur_pos>=16)&&(cur_pos<=23) )
                cur_pos=24;
        else if( (cur_pos>=24)&&(cur_pos<=31) )
                cur_pos=32;
        else if( (cur_pos> 32)&&(cur_pos<=39) )
                cur_pos=40;
        else if( (cur_pos>=40)&&(cur_pos<=47) )
                cur_pos=48;
        else if( (cur_pos> 48)&&(cur_pos<=55) )
                cur_pos=56;
        else if( (cur_pos>=56)&&(cur_pos<=63) )
                cur_pos=64;
        else if( (cur_pos>=64)&&(cur_pos<=71) )
                cur_pos=72;
        ++i;
}

                                        /* To discard load values on monitor screen. */
else if( ((g>=48)&&(g<=57)) || (g==46) )
{
        ++cur_pos;
        ++i;
}
else
        ++i;
}                                       /* end while. */
}
                                        /* This routine take control character and adjust
                                        the index value inside the screen line data
                                        structure. */
void ctr_char(int control_char,int sys,int user,int line,int flag,
int val,int g)
{

if(control_char==0)
        control_char=C;


if(val==0)
{
        if(vflag)
                val=0;
        else
                val=1;
}


switch(control_char)
{
        case C:                         /* If control character is C then go to forward. */
        fward(C,sys,user,line,flag,val,g);
        break;
```

```
        case D:                         /* If control character is D then go to backward. */
        backward(D,sys,user,line,val,g);
        break;
        case A:
        break;
        case B:
        break;
        case H_CONT:                    /* If control character is H then go to new line. */
        new_line(H_CONT,sys,user,line,val,g);
        break;
}
vflag=0;
}
                                        /* This routine get the numerical value from each
                                        monitor output line. */
int value(int i,int n,char *item)
{
int p=0;
        if(i>(n-1))                     /* If index is greater than string length then it is
                                        error. */
                return(-1);
        p=(int)item[i];                 /* Get integer value. */
        return(p);
}


                                        /* This routine converts ASCII number into integer
                                        number. */
int convt(int j,int k)
{
int T[2];
int i,cn;

        T[0]=j;
        T[1]=k;
        if(T[0]==-1) return(1);
        for(i=0;i<2;i++)
        {
                switch(T[i])
                {
                        case 48:        /* If the ASCII value is 48 then return 0. */
                        cn=0;
                        break;

                        case 49:        /* If the ASCII value is 49 then return 1. */
                        cn=1;
                        break;

                        case 50:        /* If the ASCII value is 50 then return 2. */
                        cn=2;
                        break;

                        case 51:        /* If the ASCII value is 51 then return 3. */
                        cn=3;
                        break;

                        case 52:        /* If the ASCII value is 52 then return 4. */
                        cn=4;
                        break;

                        case 53:        /* If the ASCII value is 53 then return 5. */
                        cn=5;
                        break;

                        case 54:        /* If the ASCII value is 54 then return 6. */
                        cn=6;
                        break;

                        case 55:        /* If the ASCII value is 55 then return 7. */
                        cn=7;
                        break;

                        case 56:        /* If the ASCII value is 56 then return 8. */
                        cn=8;
                        break;

                        case 57:        /* If the ASCII value is 57 then return 9. */
                        cn=9;
                        break;
```

```
                        default:        /* Default value is -1. */
                        cn=-1;
                        break;

                }                       /* end switch */
                 T[i]=cn;
        }                               /* end for */
        if(T[1]!=-1)
                T[0]=( T[0]*10 )+T[1]; /* Multiply by its weights. */
        return(T[0]);
}
                                        /* This routine positions the cursor to upward
                                        direction. */
void upward(int val)
{
        if(line !=0 )
                line=abs(line-val);
        return;
}

                                        /* This routine positions the cursor to backward
                                        direction. */
void downward(int val)
{
        line=line+val;
        return;
}


                                        /* This routine positions the cursor to forward
                                        direction. */
void fward(int control_char,int s,int u,int Ln,int flag,int val,int g)
{
int indx;

        if(control_char==C)
        {
                if(flag)
                {
                        val=val-9;
                        if(val==0)
                                cur_pos=1;
                }
                else
                        cur_pos=cur_pos+val;
                update_sline(s,u,Ln,g);
                s=0;
                u=0;
                flag=0;
        }
}
                                        /* This routine positions the cursor to backward
                                        direction. */
void backward(int control_char,int s,int u,int Ln,int val,int g)
{
int pos,indx;
        if(control_char==D)
        {
                if(flag_ok)
                        val=val+9;
                cur_pos=cur_pos-val;
                update_sline(s,u,Ln,g);
                s=0;
                u=0;
                flag_ok=0;
        }
}
                                        /* This routine executes whenever the new line
                                        control character appears in the monitor output
                                        line. */
void new_line(int control_char,int s,int u,int Ln,int val,int g)
{
int indx;

        if(control_char==H_CONT)
        {
                update_sline(s,u,Ln,g);
                s=0;
                u=0;
        }
```

```
}
                                        /* This routine update the screen line data
                                        structure by inserting the extracted data from the
                                        monitor output and then calculate system and user
                                        time by counting the system and user time
                                        characters. */
void update_sline(int s,int u,int Ln,int g)
{
int k=1;
int user,sys,i;
static int countr_T=0;
int ct_T=23;
if(s!=0)                                /* If system time is not zero. */
{
        while(k<=s)
        {                               /* Insert the system time character (5).*/
                sline[Ln].L[cur_pos]=5;
                ++k;
                ++cur_pos;              /* Increment line index. */
        }
}

k=1;
if(u!=0)
{
        while(k<=u)
        {                               /* Insert the user time character (9). */
                sline[Ln].L[cur_pos]=9;
                ++k;
                ++cur_pos;              /* Increment line index. */
        }
}
s=0;                                    /* Reset the system and user time. */
u=0;
k=1;
if(sp_ctr != 0)                         /* If there is a space character. */
{
        while(k<=sp_ctr)
        {                               /* Insert the space character (0). */
                sline[Ln].L[cur_pos]=0;
                ++k;
                ++cur_pos;
        }
        sp_ctr=0;
}

i=1;
sys=0;
                                        /* Now count the system time character to calculate
                                        the system time. */
for(i=1;i<=60;i++)
{
        if(sline[Ln].L[i]==5)          /* For system time. */
        ++sys;
}
user=0;
                                        /* Now count the user time character to calculate
                                        the user time. */
for(i=1;i<=60;i++)
{
        if(sline[Ln].L[i]==9)          /* For user time. */
        ++user;
}
                                        /* Convert the system and user time values into
                                        percentages. */
sys=roundval(sys);                      /* For system time. */
user=roundval(user);                    /* For user time. */
                                        /* Send the line number to the parent process. */
write(p[1],&line,sizeof(line));
                                        /* Send the system time values to the parent
                                        process. */
write(p[1],&sys,sizeof(sys));
                                        /* Send the user time values to the parent process.
                                        */
write(p[1],&user,sizeof(user));
}
```

```
                                        /* This routine first rounds the system and user
                                        time values and then converts them into percentage.
                                        */
int roundval(int x)
{
float y,q,f;
int h;
        y=(float)x*10.0/6.0;
        h=(int)y;
        f=(float)h;

        q=y-f;
        if(q>=0.5)
                h=h+1;
        return(h);

}
```

```
                                        /* This routine first rounds the system and user
                                        time values and then converts them into percentage.
                                        */
int roundval(int x)
{
float y,q,f;
int h;
        y=(float)x*10.0/6.0;
        h=(int)y;
        f=(float)h;

        q=y-f;
        if(q>=0.5)
                h=h+1;
        return(h);

}
```

VITA

Syed Nasir Raza

Candidate for the Degree of

Master of Science

Thesis: REALTIME GRAPHICAL DISPLAY OF SYSTEM MEASUREMENTS

Major Field: Computer Science

Biographical:

Personal Data: Born in Karachi, Islamic Republic of Pakistan, July 8, 1966, son of Mr. and Mrs. Syed Mojiz Hussain Baqri.

Education: Graduate from Government College, Karachi, Pakistan, in May 1986; received Bachelor of Science degree in Physics from Islamia Science College, Karachi, Pakistan, in August 1989; received Master of science degree in Applied Physics from University of Karachi, Karachi, Pakistan, in December 1991; completed the requirements for Master of Science degree in Computer Science at the Computer Science Department at Oklahoma State University in December 1997.

Professional Experience: Computer Lab Consultant, Computing and Information Services, Oklahoma State University, September 1996 to December 1997; Computer Graphics Artist, The Daily O'Collegian, Oklahoma State University, January 1996 to December 1996.