XNUSIM – Graphical Interface for a Multiprocessor Simulator

Pang, Swee-Chee

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley, California 94720

Abstract

Xnusim is an X11 Window Interface for the Multi-Processor simulator Nusim. It is a display oriented interface between the simulator and the user via *UNIX*[1] *sockets* with graphical objects such as menus, buttons etc. It is designed in such a way that would allow it to be used with other simulators of the same class. This paper intends to describe the functionality of the objects, structures and program modules of XNUSIM in detail .

September 8, 1989

[1]UNIX is a registered trademark of AT&T Bell Laboratories in the USA and other countries

Acknowledgements

I would like to thank Dr. Vason Srini for his valuable advice and guidance. I would also like to thank Tam Nguyen for his input and feedback on the xnusim program.

My thanks also to Darlene Gong whose incessant urging and confidence kept me going.

This research was partially sponsored by Defense Advanced Research Projects Agency (DoD) monitored by Office of Naval Research under Contract No. N00014-88-K-0579, NCR Corporation in Dayton, Ohio, and National Science Foundation.

Contents

Section 1

Introduction

Xnusim was built with the intention of giving Nusim a more visual interface. Nusim[NC89] is a simulator for the PPP (Parallel Prolog Processor) [Fag87] which is part of the Aquarius Project, at the University of California at Berkeley[DS88]. However, aside from knowing the input-output semantics and the kinds of commands nusim accepts (refer *Section 5*), Xnusim does not require knowledge of what level simulation is performed and what kinds of details are involved in the simulator, so long as it adhere to some fixed set of criteria which will be presented at the concluding section (*Section 6*).

Due to this method of interface, xnusim should not be difficult to be converted to interface with other simulators, especially if care is taken in writng a simulator with similar debugging capabilities. *Section 5* will describe methods of interfacing with xnusim, changes that can be easily made, and will also outline the criteria for writing a compatible simulator.

Xnusim is an interface built on top of the *X Toolkit Library* [MAS89] under *X Protocol Version 11 Revision 3* [1][GSN89, SG86]. A brief introduction into the X11R3 Windowing system and the XToolkit along with some of the other software used will be presented in *Section 2.* In this same section, the *4.3BSD Communication Protocol* [LMKQ89] will also be discussed; to be specific, the use of *sockets* which is what xnusim uses to communicate with nusim.

[1]The X Window System is a trademark of MIT. Copyright ©1985, 1986, 1987, 1988 Massachusetts Institute of Technology, Cambridge, Massachusetts, and Digital Equipment Corporation, Maynard, Massachusetts.

Section 3 will present an overview of xnusim, while *Section 4* will explain the technical details that makes up the complete xnusim program set. The concluding section will discuss improvements possible or desirable. Attached as appendixes are the man page, a list of xnusim's procedures and files where they may be found and after that, a list of the entire xnusim program in C.

Section 2

General System Requirements and Overview

2.1 X Window System

The X Window System[SG86] was designed by MIT as a windowing system which runs under 4.3BSD UNIX and several other variants and has since become available for the VAX/VMS, MS-DOS and other operating systems as well. The display server is a network-transparent interface that accepts output requests from various client programs and handles user input which could be of the form of keyboard or mouse events. The client programs need not necessarily be located on the same machine. The version of X used is the X Protocol Version 11 Revision 3 System (X11R3) [GSN89]. Xnusim cannot be used with X of a lower protocol system since it makes use of certain features which had become available only in the X11R3 system. It is conceivable that it will run on later releases with minor or no changes at all.

In order to more easily implement the system, the X Toolkit[MAS89] was used. It is also believed that although much of the X11 system might be changed with latter releases, updates and bug fixes, the X Toolkit is a relatively stable application package and utilizing it instead of direct interface to the X11 system calls would render the software more lasting and less reliant on the system and the update versions.

The X Toolkit Intrinsics, redesigned for the X11R3 windowing system, is intended

to provide some basic mechanism to build sets of *widgets* for any application environment. A widget is the fundamental abstraction and data type of the X Toolkit and can be visualized as a blackbox state machine with associated input/output semantics. Some widgets display information like text or graphics while others may serve as a container for other widgets The Intrinsics is built on top of Xlib and serves as an abstract, object based extension to the X Window System. X Toolkit provides an interface which is consistent throughout, and a small set of intrinsics easily used to write applications and at the same time provides those same set of Intrinsics suitable for building other widgets. Because of the way the Intrinsics is designed, constructing other widgets is almost trivial.

In writing xnusim, extra widgets such as the "Scroll" and "MenuBox" widgets were constructed and used along with the basic X Toolkit Intrinsics. Documentation for these two widgets are available as part of the distribution for these new widgets, or may be found, respectively, in the subdirectories "Scroll" and "MenuBox" under the "xnusim" directory.

2.2 UNIX 4.3BSD Communication Protocol

One of the many features in UNIX 4.3BSD is that of interprocess communciation (IPC)[LFJ+86, LMKQ89]. It provides capabilities from network level to process level communications via relatively simple and transparent means. The 4.3BSD IPC allows different processes to communicate via many different ways and levels.

For the purpose of xnusim, communication was needed between that of xnusim and the nusim simulator. Nusim was designed primarily without considerations of whether a higher level interface was available and used, and takes it's input and output from the terminal. Since one of the goals of xnusim was to provide an interface that was invisible to the simulator as well, the most appropriate means of communication was thought to be that of *pseudo terminals*. The pseudo terminal model has two parts: a master and a slave terminal part.

The main process, for example, *xnusim*, may send data, in our example, this could be a command to nusim, through the master side which will be passed to the slave

"terminal" as *stdin*. Any process (nusim) which exist at the slave end will then be able to pick this data up as normal standard input. Similarly, the process at the slave end may output to either standard error or standard output (*stderr* and *stdout* respectively) and these will be picked up at the master end as data from the slave and may then be processed accordingly (like output into the main window etc).

Using this method of communication, nusim is completely oblivious to the existence of a process image of xnusim executing above and controlling it.

Section 3

Overview of Xnusim and User Reference

3.1 Design Considerations

Xnusim was designed as an interface to nusim, but it was also desired that xnusim be sufficiently flexible to be easily adapted to other simulators. Therefore, an interface that was *loosely coupled* to the simulator was decided upon. Loosely coupled in the sense that the simulator has no knowledge of the existence of xnusim, and xnusim has little knowledge of the workings of the simulator. And what little xnusim needs to know about nusim in order to function was localized into specific parts, so as to minimize the modifications necessary to allow it to function with other simulators.

Figure 3.1 is a simple construction of the visualization of the design consideration for xnusim. In the figure, xnusim communicates with the user via the X11R3 window system, through the use of menus, command buttons, and keyboard entries. All these are processed by the window system before passing down to xnusim. Xnusim communicates with the simulator (through IPC) in such a fashion that the simulator thinks it is in direct communcation with the user.

This method of communication gives the most flexibility to xnusim and also frees the programmer of the actual simulator (nusim) from needing to put the interface into consideration when designing the simulator.

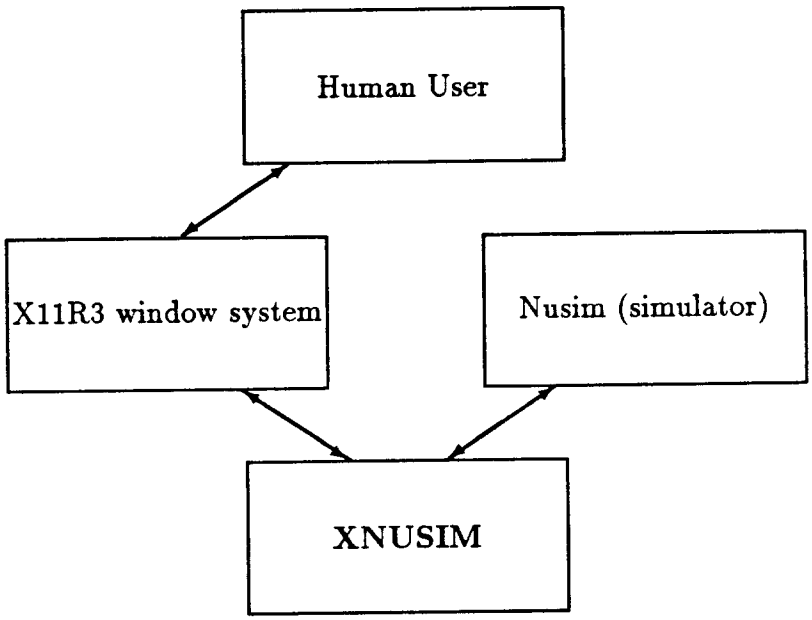

Figure 3.1: XNUSIM's communication virtual view

The main objective of xnusim is to provide a graphical interface which is capable of supporting a parallel processor simulator and give the user a visual and easy to understand mouse-menu oriented system. The behavior of the simulated programs can be studied by observing the processors/tasks displayed by xnusim. Therefore, the capability of displaying information for multiple processor and tasks was necessary. But the user must be given an option to choose the number and which of the processor/task(s) to display at will since the use of single screen display limits the amount of information possible (xnusim can be easily reconfigured to display on multiple screens).

3.2 Windows

Xnusim is a window oriented display, and manages several windows, which are, technically speaking, actually widgets. And for the purpose of this section they will be used interchangeably unless specifically mentioned otherwise, due to subtle technical differences. Upon startup, a large window appears which contains several subwindows, menu-windows

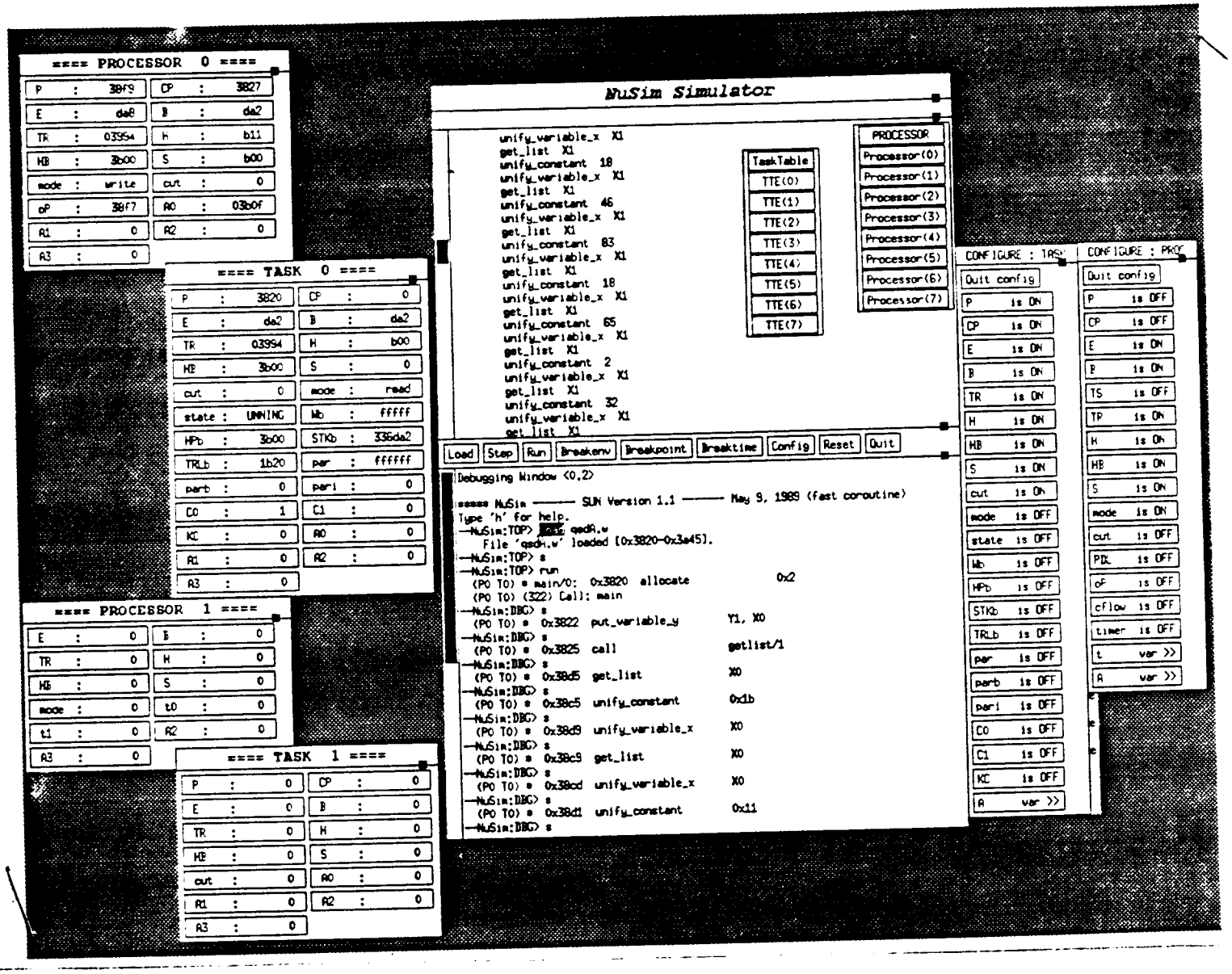

Figure 3.2: XNUSIM's screendump of all stable windows

may appear on request and also windows for configuration and a window each for individual task/process that the user chooses to display. The following subsections will discuss each of the type of windows. Figure 3.2 shows a diagram of most of XNUSIM's windows, and it is suggested that this be used for cross-referencing the description to follow. In this figure xnusim's "stable" windows are displayed. By stable windows, it is meant that the windows will not disappear the moment the mouse leaves that window. The step sized has been set to 2 in the figure as can be noted by comparing the step display in the main debugging window and the listing window.

3.2.1 Main Window I: Titlebar

The titlebar widget shows the title currently assigned to nusim (easily changed in "defaults.h" as *"SimulatorName"*), but also serve as the sensitive point for starting up of the main menu which allows the display of processors and tasks.

3.2.2 Main Window II: Helpbar

The help widget simply display any error message or messages explaining the use or name of the window that the mouse is in.

3.2.3 Main Window III: Listing Window

This window is where the program(s) being simulated is loaded into. There is a cursor in the window which will always be updated to point to the current instruction being executed after each "step" or "run" instruction. The user may reposition the cursor anywhere and then set breakpoints at the position where the cursor is (refer 3.2.4). Nested (or include) files are listed one after the other in the window, in the order by which the simulator returns them.

3.2.4 Main Window IV: Command Window

The command window consists of several command buttons, and all these commands may be activated by pressing the left mouse button (unless otherwise reconfigured)

on that command button. Below is a short description and explanation of the command buttons as they appear in xnusim.

Load Pressing this button will create a dialog widget where you may enter the filename of the byte compiled program which you wish to simulate.

Step Pressing this button makes xnusim step the simulator n times where n may be configured under the **config** option (see below).

Run Pressing this button for the first time sends the "run" command and subsequently it will send the "c" command (for continue) which will cause the simulator to run itself until the end, an error or a stop point. Pressing **reset** (see below) will cause it to send the "run" command the first time this button is activated after that.

Breakenv A dialog window with two inputs, one for process environment, and the other for task environment, will pop up and the user may change them. A return key at either input line ends this function.

Breakpoint A menu listing whether the user wishes to select setting trace/break points at the current cursor position or wishes to input his own trace/break points and a list of all deletion options currently available will be displayed. Of the list of options offered, if no breakpoints were set, the list of deletion options is empty; if only one break/trace point was set, the list has only that member; and if more than one were set, the list has the "delete all" option as well.

Breaktime A dialog window will be available to set the breaktime (or delete it).

All three are updated at the point of pressing the button, so the user may set/change these on the main debugging window (refer Subsection refdebugwin) and the updates will be available here as well.

Config This button activates the config window which currently contains 3 parts:

- *Step* Where a dialog window will pop up for the selection of the number of steps which the step button will perform.

- *Processor* A configuration list of all known registers for the processor module will be listed with their current display status (ON : display; OFF : not dislayed) or, if they're variable (*eg* A[0-7]), the arrow in place of the ON/OFF display will indicate that going there will make another window pop up showing which of the variable number (MAXNUM set in "processor.h") register is being displayed.

  The user may press on these button to update the display status of that register. Update is instantaneous and the user may leave this window active while selecting a new processor to display. As a policy decision, processors already being displayed will not have these update affect them. In reference to figure reffull-window1 Processor 0 and Task 0 in the figure were activated with the default registers selection and Processor 1 and Task 1 were activated after the set up change (compare with the "Configure" windows on right side of the figure which displays the register setup for the new processor and task and not the default).

- *Task* Similar to the processor module.

Reset Terminates nusim and restarts it. This allows the user to be able to start with a clean copy of nusim without the need to quit xnusim and then re-setup the task/processor and other display features.

Quit Simple enough: quits xnusim.

3.2.5 Main Window V: Main Debugging Window

This window is where the user will see the bulk of the activity occur. The communication between xnusim and nusim will be displayed here, and the user may edit and type in line commands to nusim directly from here too.

Section 4

Technical Details: Layout of Xnusim

4.1 Introduction

Xnusim is made up of and 2 widget library files and 14 files, 7 of which are "header" (".h") files, The library files have their own description and references, so this section will be mainly describing the 14 files. The names of procedures used in xnusim are shown in *Appendix A*. The manual page for xnusim is found in *Appendix B*. The actual listings of the 14 files are in *Appendix C*. Of these 14 files, 2 of them, *general.c* and *general.h*, are files which are useful for any program since commonly needed routines are placed there.

4.2 Descriptions of Individual Files

- **general.c and general.h**

The two files define the general routines that may be used for almost any application. Routines there maybe found in any good C book. Included are definitions for CALLOC, MALLOC, LARGE and *forever* which speak for themselves, *min* and *max* which return the larger/smaller of two, *error* which prints an error message and may quit if desired, *inchr* and *instr* which checks if a certain charactor/substring is in another string, and *hextoi* and *itohex* which converts between hexadecimal numbers and decimal numbers.

- **defaults.h**

In this file is all the default names and sizes used by xnusim, and would probably be changed by the user when porting and re-adapting xnusim for other purposes. This file is needed by all the other files to get their default sizes, fonts and name used.

- **interface.h**

The file which is definitely sensitive to the kind of simulator used. Defined in here are the types of commands recognized, what is a PROMPT, and the functions available for general use by other files.

- **mainmenu.h**

Defines the window information and callback functions for the main menu (refer Section 3.2.1).

- **manager.h**

Basic definitions for Xtoolkit functions.

- **menucmd.h**

This is the Window counterpart to *interface.h.* It defines the commands which appear in the command window (refer Section 3.2.4) and the functions to call (in *handler.c*) when that command button is activated[1].

For both menu windows (main menu and command window), there is a help information which is displayed whenever the mouse enters that button. This help information is displayed in the help window (refer Section 3.2.2).

- **processor.h**

[1] A button is termed "activated" when the mouse is placed at that button widget and the activation button, normally the left mouse button, is pressed.

This should be more appropriately called *processor_and_task.h*, but this name was chosen as it is sufficiently long without being awkward. This file defines the maximum processors and tasks registers and what they are, and also defines the number of variable number register[2]. It also defines the default registers of the entire set which is activated.

- **handler.c**

A common module for any Xtoolkit application program. All the functions that are called when the commands and menu buttons on xnusim are activated are described here. This probably needs to be modified whenever the commands are changed, but modifications could be simply cut and paste since most forms of buttons are available, and any programmer sufficiently versed in C and X11 will immediately recognize the order of changes. Most of these makes calls to the *interface.c* module (most probably via the *sendMsg* procedure) which does most of simulator dependent work. Most likely to change are the "Break" series of buttons since these were made specifically for nusim. But it was deemed necessary. This module has to be changed when it becomes desirable to interface xnusim with other simulators.

- **interface.c**

All of the simulator dependent functions are found here (except for those related to processors and tasks some of which may be found in the *misc.c* file). A more detailed discussion of some of the functions in this module is in order and the user is referred to *Section 5* for that. This file is the crux of the interface between nusim and xnusim. All of xnusim's calls from the user eventually ends up to some routine in this file. There is a routine (*MainDo*) which will recognize nusim's output and calls the appropriate routine (most probably also in this file) to update it's values, like the listing window (on load and step/run) and the processor/task windows (*misc.c* involved). It is possible to drop *misc.c* and attach these functions here, but it was decided to localize all processor/task related function to a file.

[2]For the purpose of this paper, a "variable number register" is a register with suffixes from 0 to a maximum number defined in that file, like the "A" register which may have suffixes from 0 to 7 thus "A0"–"A7"

- **main.c**

Does the initial command line interpretation, performs the necessary "forking" of processes and executes each correctly. Trap for exit errors is also found in this file. The user is referred to the xnusim's manual page for the list of options available.

- **manager.c**

This is the main file for interfacing to Xtoolkit. It does the initial and main graphics set up for xnusim, defines each window, and their components and then display them. It also starts the infinite loop that executes xnusim's part of the Xtoolkit interface.

- **misc.c**

This file defines all of the modules needed for the processor and task subwindows. The processor and task windows are similar in nature, merely differing in names and actual register set. Thus, modifying one would imply modifying the other (refer to *Section 5* for details on modification). The file contains the functions which pop up each processor/task window, the functions called when the values need to be updated, and the functions called when there is some configuration necessary for the register sets for the processor/task windows.

4.3 Interaction of Xnusim's Modules

To understand the interaction between these modules (files), the user should get familiar *Appendix A* that lists the functions, and which files contain these functions. To give a general view of the module's interaction, consider when the user types in a command or presses a button. The eternal loop in *manager.c* captures that "event"[3], then the related functions are called.

The key events are now described:

[3]events are any form of action related to the widgets, including exposure, keyboard input, mouse input, size change etc

- If the event is a keyboard input in main window, these functions are found in *manager.c* which is called and then returned to the eternal loop (*forever* line), unless the "return" key is hit, whereby the keyboard interpretation function in *manager.c* will call *interface.c* which will transmit that command to nusim, and then the eternal loop will be returned.

- If the event was that of a button pushed, then the functions in *handler.c* will be called which eventually (perhaps after some menu which are found in *handler.c*) will call *interface.c* which will again transmit that command to nusim, and then return to the eternal loop (in *manager.c*). If, however, this button was to perform some function with task/processor windows, the file *misc.c* will be called instead of the *interface.c*. Besides configuration, however, *misc.c* will eventually also call *interface.c*.

If there is any output from nusim, then as part of the eternal loop, the *MainDo* function in *interface.c* is called. Here, the function will detect the reply, does some simple interpretation and then pass it on to the appropriate functions in *interface.c*. When these functions return, it will then call the processor/task windows to update the appropriate table. Note that these will be done *iff* an output from nusim is expected.

A detail missing from the description is that whenever read and write is performed, the functions *MessageRead/Write* of *main.c* will be eventually called which does the raw block transfer between xnusim and nusim. These are *NOT* simulator dependent since they merely transfer the raw bytes from the master terminal to the slave terminal and vice versa.

Section 5

Interfacing Xnusim to Other Simulators

5.1 Introduction

As xnusim was designed, it was decided that a desirable feature would be to make xnusim sufficiently general that it would be easy to modify it to work with other simulators. Therefore xnusim was designed so that it made as little assumption on the way the simulator performs as possible. Also due to this, the simulator dependent functions have been localized to only a few modules. This section intends to outline these modules and methods of modifications that would allow xnusim to work with other simulators that adhere to the assumptions listed below.

- The simulator is assumed to have at most multiple procesesors and tasks of the same class, ie, all processors are homogeneous in terms of register sets, and similarly for tasks. In this class of simulator is included those simulators which have single task and single processor and those with either multiple tasks or processors which are homogeneous.

- Upon receiving any command, the simulator is assumed to output some feedback messages which always end with some predefined prompt. This feedback scheme is necessary only so that xnusim may perform updates correctly, while the predefined

prompt is used by xnusim to recognize that nusim has stopped sending output. For this reason, the simulator would need to have some fixed number of prompts to function properly.

- The simulator is assumed to need to load some source file which is in ascii format. Of this loaded format, it is assumed that the simulator will deal with the simulator at that level as well (It may or may not deal with other levels of coding). This is required to ensure that the listing window will perform some useful update with the source code that is loaded. Nested files and/or include files can be handled as well.

- It is also assumed that the simulator has command(s) that will enable xnusim to enquire about the status of the processor/tasks registers, current simulator position in source code, break points set.

Of course, these may or may not remain valid depending on the level of changes made to xnusim, but the simplest changes are necessary for those simulators adhering to the criteria given. The following section will discuss specifically how to modify xnusim to interface to simulators agreeing with those above.

5.2 Modifying The Interfacing Module

There are basically three things that need to be modified in xnusim to interface to the new simulator. The first is the way xnusim interpretes an output from the simulator, since it is expected that the simulators would definitely defer there. The modifications will be localized in the file *interface.c* in this case, and some changes to the file *misc.c*. The second is the names of registers for processors/tasks. This is only in *processor.h*. The third is the command buttons and the way they are handled. This is in the module *menucmd.h* and *handler.c*.

5.2.1 Simulator Communication

Most of the simulator interpretation is located in just one file, *interface.c*. The only other file is *misc.c* which has *two* procedures (one in *updateTask* and the other in

updateProc) that are dependent to simulators.

The two procedures in *misc.c* are images of each other, following the philosophy of treating tasks and processors similarly in this simulator, so description of only one is necessary. The procedure ***updateTask*** first sends a command to the simulator to print out the current register condition for the specific task. The simulator's output is assumed to be of the form[1]:

```
{({<SPC>*<REG>':'<SPC>*<VAL><SPC>*}*<rubbish>*)*'\n'}*
```

If the simulator output differs, then this procedure will have to be modified.

The file *interface.c* is where the major changes would be required. (Remember to change *interface.h* if necessary) Below is a quick discussion of most of the procedures, the rest would be self-evident after these.

needline: Probably would not need to be changed unless there is a change in which the interpreter is supposed to perceive an "end of output stream" from the simulator, which currently is when it reads a line ending with the PROMPT. It returns a line that is read each time.

doload: Needs to change only the part which sends the "load" command *iff* the simulator does not accept the command sequence of "load filename".

loadprocess: Parses through the *buf* variable passed (raw bytes read in). It assumes the buffer to be of the form:

```
{<rubbish>'\''<FILENAME'\''<rubbish>'['<SPC>*<ADR><SPC>*'-'<SPC>*<ADR><SPC>*}*
```

where ADR is assumed to be a hex address (see procedure *gethex*) and the content is assumed to be the filename and the starting address and ending address of the file as it is loaded in memory. (This probably would need changing for another simulator) Once it gets the filename, it loads the file into the listing buffer, while updating the count of number of lines and where each line is in the charactor array that makes up the listing buffer. The loading part do not need to be changed. Next it tells the simulator to list it's version of the code, and then try matching it according to the file it loaded. It assumes the list to be of the format:

```
{<ADR><SPC>*':'<SPC>*<CODE><rubbish>'\n'}*
```

[1]Expressed as a regular expression, where SPC is white space, REG is register name, and VAL is value of register

And will then match the lines according to this listing, line by line. It thus assumes the simulator will *NOT* modify the code as it is loaded. If the simulator does so, xnusim will run, but will not be able to update the listing window pointer accurately and may produce unpredictable results.

updateenv, updatebreaktm: These are also reliant on simulator and are quite similar, assuming the same command in the simulator will provide information for both, but on different lines. Code is simple enough to understand.

updatebreakpt: This assumes the first line would have a ':' if there had been any breakpoints set, otherwise it returns. Simulator should output breakpoints of the format:

```
{<digit>+'.'[(<rubbish>':')U()]<ADR><SPC>*'('['b'U't']')'<rubbish>'\n'}*
```

Where address is the hexadecimal address of where the breakpoint is set, and the 'b' or 't' charactor indicates whether it is a break or trace point. This module probabley needs to be changed for other simulators.

sendMsg: The function which is most important in communicating to the simulator. Does multiple command communication to the simulator. For each command, it sends the command and then returns. For some commands it sends the command multiple number of times.

MainDo: This function is the loop that will read an output from the simulator if it is expected, and assumes there will be no more output for the time when it sees the PROMPT, and will also branch to the *loadprocess* and *updateProc/Task* procedures. It also respositions the listing window if it detects movement in the pointer in the simulator. Therefore, it is necessary to have the simulator output some address information if there is to be consistent update for the listing window with the actual stepping of the simulator.

The changes in *misc.c* and *interface.c* will not affect the execution of other parts if the information returned and variables accepted are the same. It is believed that regularity and special keyword output from the simulator would make *interface.c* module relatively simple.

5.2.2 Register names

The file that needs to be changed is *processor.h*. For the purpose of xnusim, two kinds of registers are distinguished. The normal ones and those with variable number, like A[0-7] for nusim. The constants which control the number of registers and the number of variable registers are self-documented in that file. The names of each register for processor are in *proc* and those for task are in *tte*, both of which are charactor string arrays. Merely type in the names (remember to change MAXLEN if there are reasons to use registers name with more charactors than those defined there) in double quotes.

The variables *procstat* for processors and *ttestat* for tasks define the initial display information for xnusim's processor/task set. They define whether the corresponding register defined in *proc* or *tte* is, by default, being displayed, not being displayed or a variable register type. If it is the variable type, the number indicates the index (+1) into the corresponding *procvar* or *ttevar* arrays where the same displayed or not displayed information, as applied to variable registers, may be found.

5.2.3 Buttons

The last thing that probably needs to be changed is the *handler.c* module which handles the button responses. For each button that is changed, there is probably need to change the *menucmd.h* file which contains the names of the buttons and the functions they call. The comments in *menucmd.h* would be sufficient to modify that file. In order to modify the file *handler.c*, some knowledge of Xtoolkit is necessary. Since only basic functions like *XtSetValues*, *XtPopup*, *XtAddEventHandler* etc are used, basic knowledge of Xtoolkit and X11 system would be sufficient to understand and modify this module.

Section 6

Conclusion

6.1 Summary

This paper outlines the entire project for Xnusim, which started as a simple interface for a simulator under development at that time but developed into a general debugger interface. The paper covered the areas of what xnusim is, how xnusim is designed, what to modify when changes are needed, and what kind of support xnusim gives to and requires from the simulator.

Xnusim would definitely provide an environment that will ease the user from the need to keep track of several processors and tasks, and would make it easier for the user to debug the source code and understand how the parallelism functions because it displays most of the essential information via windows and allow the user to perform several tasks via simple button clicking.

Xnusim has been shown to be a powerful interface tool for simulators. Writing a simulator that is graphics in nature limits its used to that graphics environment. Writing a simulator without graphics capability makes studying parallelism and debugging source code a cumbersome process. Thus, xnusim serves as a solution to this seeming conflict. The simulator may still be used in non-graphics environment or any environment of a different nature, but when desired, xnusim will serve as the graphical link which will solve the second part of the problem.

6.2 Future Development

Many improvements are possible to xnusim. Some of them are outlined below.

I Xnusim should become much more user friendly, for example, the "loading" (which could perform directory listing) command.

II Xnusim's interface to the simulator could be improved, for example, listing of breakpoints in the listing window.

III There is currently no summary information printed by xnusim. This is a definitely desirable feature to be included. But it has not been included since what kind of information and how these informations are to be arranged and gathered has not been well-defined.

IV The module *handler.c* may be modified to be sufficiently general that it will become unnecessary to modify it for any modification to the simulator. This is possible if a protocol for defining what kind of menus, how these are to be manipulated and what functions they call is established. Then, the main function for interpreting this will be *handler.c*'s heart, and possibly the procedure *sendMsg* of *interface.c* would become more sophisticated.

V The next giant step would be to make *interface.c* a general file that does some form of regular expression interpretation and replies with some regular expression, all of which may be defined, again, by some protocol. If this is done, using a *configuration* file of some sort for the kind of simulator, xnusim would be able to handle different simulators without ever needing any recompilation, and would truely establish the ideal of being a general simulator interface. (Incidentally, this would include the modifications mentioned for the *handler.c* module, since it would not work otherwise)

With these modifications, xnusim would probably be a very useful package for people interested in designing parallel systems at different levels, debugging programs that are to be used in these systems, and studying the behaviour of different programs.

Bibliography

[DS88] A M Despain and V P Srini. Aquarius Project Technical Progress Report, DARPA Contract No. N00014-88-K-0579. Technical report, October 1988.

[Fag87] Barry S Fagin. *A Parallel Execution Model for Prolog.* PhD thesis, CSD, University of California, Berkeley, November 1987. Report No UCB/CSD 87/380.

[GSN89] James Gettys, Robert W Scheifler, and Ron Newman. *Xlib - C Language X Interface, X Version 11, Release 3.* Massachusetts Institute of Technology, 1989.

[LFJ+86] S J Leffler, R S Fabry, W N Joy, P Lapsley, S Miller, and C Torek. An Advanced 4.3BSD Interprocess Communication Tutorial. *UNIX Programmer's Manual, CSRG*, page PS1:8, April 1986.

[LMKQ89] Samuel J Leffler, Marshall K McKusick, Michael J Karels, and John S Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System*, chapter 10. Addison-Wesley Publishing Company, 1989.

[MAS89] Joel McCormack, Paul Asente, and Ralph R Swick. *X Toolkit Intrinsics - C Language Interface, X Version 11, Release 3.* Digital Equipment Corporation, 1989.

[NC89] Tam M Nguyen and Chien Chen. A simulation system for multiprocessor architectures. Technical report, Aquarius Project Technical Progress Report, DARPA Contract No. N00014-88-K-0579, April 1989.

[SG86] Rober W Scheifler and James Gettys. The X Window System. *ACM Transactions on Graphics*, 5(2):79–109, April 1986.

Appendix A

Procedure Listing for Xnusim

Procedure Name	File of origin	Prototype of Procedure
ClrSel	manager.c	XtActionProc ClrSel(w, event, parm, num)
DelChar	manager.c	XtActionProc DelChar(w, event, parm, num)
DelLine	manager.c	XtActionProc DelLine(w, event, parm, num)
DelWord	manager.c	XtActionProc DelWord(w, event, parm, num)
Killconfig	misc.c	void Killconfig(w, client, call)
MainDo	interface.c	void MainDo()
MessageRead	main.c	int MessageRead(buf, n)
MessageWrite	main.c	int MessageWrite(buf, type)
Mmain	main.c	main(argc, argv)
ModifyProcReg	misc.c	void ModifyProcReg(w, client, call)
ModifyTaskReg	misc.c	void ModifyTaskReg(w, client, call)
ModifyVarReg	misc.c	void ModifyVarReg(w, client, call)
SelWord0	manager.c	XtActionProc SelWord0(w, event, parm, num)
SendCmd	manager.c	XtActionProc SendCmd(w, event, parm, num)
SetVarReg	misc.c	void SetVarReg(w, client, call)
SigInt	manager.c	XtActionProc SigInt(w, event, parm, num)
bombed	main.c	bombed(sig, code, scp)
breakenv	handler.c	void breakenv(widget, client, call)
breakpoint	handler.c	void breakpoint(widget, client, call)
breaktime	handler.c	void breaktime(widget, client, call)
buttons	handler.c	void buttons(widget, client, call)
config	handler.c	void config(widget, client, call)
configProc	misc.c	void configProc(sendtop)
configTask	misc.c	void configTask(sendtop)
control	handler.c	void control(widget, client, call)
dialog	handler.c	char *dialog(str)
dispbreakpt	handler.c	dispbreakpt(widget, j, call)
dispbreaktm	handler.c	void dispbreaktm(widget, i, call)
displayprocess	handler.c	void displayprocess(widget, i, call)
displaytask	handler.c	void displaytask(widget, i, call)
dispsize	manager.c	void dispsize(size)
dobreak	interface.c	int dobreak(linenum, mode)
doload	interface.c	static void doload()
error	general.c	error(str, type)
findLine	interface.c	int findLine(position)
findplace	manager.c	int findplace(str, posn)
format	misc.c	static void format(label, name, val)
gethex	interface.c	int gethex(s)
getlistposn	manager.c	int getlistposn()
getport	main.c	void getport()

Procedure Name	File of origin	Prototype of Procedure
handler_init	handler.c	void handler_init(pass)
help	handler.c	void help(widget, text, event)
hextoi	general.c	hextoi(str)
inchr	general.c	inchr(str, c)
init_interface	interface.c	void init_interface(size)
instr	general.c	instr(s1, s2)
interface_init_screen	interface.c	void interface_init_screen(scr1, scr2, scr3)
itohex	general.c	char *itohex(val, size)
killChild	main.c	killChild()
killWindows	handler.c	void killWindows()
load	handler.c	void load(widget, client, call)
loadprocess	interface.c	void loadprocess(buf)
makemenu	manager.c	static void makemenu(top , name)
manageProc	misc.c	void manageProc(n, top)
manageTask	misc.c	void manageTask(n, top)
manager	manager.c	manager(title, file, argv, argc)
needline	interface.c	char *needline(type)
printHelp	main.c	printHelp()
procMain	handler.c	void procMain(widget, client, call)
putList	interface.c	int putList(str, type)
putList2	interface.c	int putList2(str, type)
putMain	interface.c	int putMain(str)
quit	handler.c	void quit(widget, text, event)
reposition	interface.c	void reposition(line)
reset	handler.c	void reset(widget, text, event)
resetmanager	manager.c	void resetmanager()
run	handler.c	void run(widget, client, call)
sendMsg	interface.c	void sendMsg(sendcomm, str, times)
setdisp	manager.c	setdisp(cmd, dpy)
startsplit	main.c	void startsplit()
step	handler.c	void step(widget, client, call)
summMain	handler.c	void summMain(widget, client, call)
taskMain	handler.c	void taskMain(widget, client, call)
updateProc	misc.c	void updateProc(n)
updateTask	misc.c	void updateTask(n)
updatebreakpt	interface.c	updatebreakpt(bp, count)
updatebreaktm	interface.c	updatebreaktm(bt)
updateenv	interface.c	updateenv(task, proc)

Appendix B

Manual Page for Xnusim

NAME

xnusim – X window interface to a multiple processors/tasks simulator

SYNOPSIS

xnusim [*-toolkitoption ...*] [*-m host:display*] [*-p host:display*] [*-t host:display*] [*-s simulatorname*] [*w-filename*] [*-e simulator_options*]

DESCRIPTION

Xnusim is a graphical interface to a multiple processor and task simulator, currently implemented for the simulator *nusim*, but could be modified to handle other simulators with similar needs. It provides visual feedback and mouse input for the user to interface into the simulator.

Xnusim provides windows for each processor (maximum configurable) and task which the user wish to see, and these are updated each time the simulator returns from it's tasks.

The *-mpt* options are used to describe the display where each of the main, processor and tasks windows will be displayed (respectively).

The *simulatorname* option allows the user to specify another simulator to run under *xnusim*. However, reprogramming is necessary to support other kinds of simulators. So, this feature, thus far, only allow for name changes.

The *w-filename* option allows the user to specify a default working file which may be passed to the simulator to load once the program is started up.

The *-e* option should be the last option. Xnusim treats all arguments following this option as argument to pass to the simulator Besides these, *xnusim* accepts all of the standard X Toolkit command line options (see *X*(1)), but is yet unable to understand the simulator's options.

Xnusim is made up of the following subwindows:

Title Bar	Display the current simulator name. Also, when a mouse is place in this window, it triggers the *MainMenu*(see Below).
Message Window	Display any short Help message available and or messages from *xnusim* to the user.
Listing Window	Display the file that is currently being executed, and shows the last line that had been executed when stepping through.
Command Window	Provide a list of the commands which *xnusim* understands and is capable of executing. This is also modifiable.
Main Window	This window provides the actual simulator feedback and the user is allowed to type directly any command to the simulator through this window (Note: update MIGHT not be properly performed in that case).
MainMenu	Activated by the mouse entering the *"Title Bar"* region, it allows the user to choose to display/delete a processor or a task from the menu.

The relative sizes of any window in this set can be adjusted to suit the users needs. Although the default size is normally suggested. To select any command in a button-box, click the left mouse button.

Scrollbars can be found in both the Main and Listing windows. The left mouse button scrolls the text forward, the right scrolls backward and the middl mouse button selects the text at the current mouse position of the complete text relative to the scroll bar, changing the thumb position of the scrollbar. Dragging the middle mouse button moves the thumb along and changes the text displayed. The amount of scrolling depends on the distance of the pointer from the top of the scroll bar (or bottom). Top line scrolls one line, and bottom one screenful. Clicking the left button twice quickly on either the main or listing windows will select a word from the window which you may then echo back by clicking middle mouse. Typing a command into the debugging window will create the same effect as clicking the

mouse window.

COMMAND BUTTONS

Main Menu Commands

Processor Another window with a list of processor will popup, and choosing the processor from this new window will either delete it if it's already being displayed, or create a new window for this processor clicked.

Task Same function as the *Processor* command but for tasks.

Summary To be implemented: will display necessary statistics for the system.

Commands in Command Window

Load Prompts for the filename and then loads the ".w" file. Can only be activated once because of simulator limitations.

Step Steps throught the simulator "n" steps a time where n is defined at the *Config* button (see Below).

Run Either starts or performs continuous execution (Note: the display will not be updated).

Breakenv Prompts for new values for the processor and task break environment (see Nusim reference).

Breakpoint Allows user to delete, and set breakpoints (could set at current cursor point in listing window, program will search for first "stoppable" code memory for inserting the stop.

Breaktime Allows setting and resetting of the breaktime.

Config Allows reconfiguration of a number of things. Pressing it pops up a new window where user can select the particular type to configure.

Reset Resets the system so that you may re-run the simulator without need to exit the system. Since the simulator is actually re-runned, the whole system is completely refreshed. The only window which is not affected is the main (debugging display) window which merely reprints a start up line after the last line. This is so that you may click from the lines above to copy down.

Quit Exits *xnusim*.

LIMITATIONS

Xnusim is still underdeveloped. Much needs to be done.

BUGS

Probably quite a lot. Still shaky because of inherent problems with socket communications and Xt11.

COPYRIGHT

AUTHOR

Pang Swee-Chee, University of California.

Appendix C

Listing of Xnusim

Makefile, page 1

```
DEST        = .

EXTHDRS     = /usr/include/X11/AsciText.h \
              /usr/include/X11/Box.h \
              /usr/include/X11/Cardinals.h \
              /usr/include/X11/Command.h \
              /usr/include/X11/Composite.h \
              /usr/include/X11/Constraint.h \
              /usr/include/X11/Constraint.h \
              /usr/include/X11/Core.h \
              /usr/include/X11/Dialog.h \
              /usr/include/X11/Form.h \
              /usr/include/X11/Intrinsic.h \
              /usr/include/X11/Label.h \
              /usr/include/X11/Load.h \
              /usr/include/X11/Scroll.h \
              /usr/include/X11/Shell.h \
              /usr/include/X11/Simple.h \
              /usr/include/X11/StringDefs.h \
              /usr/include/X11/Text.h \
              /usr/include/X11/VPaned.h \
              /usr/include/X11/Viewport.h \
              /usr/include/X11/X.h \
              /usr/include/X11/Xatom.h \
              /usr/include/X11/Xlib.h \
              /usr/include/X11/Xmu.h \
              /usr/include/X11/Xos.h \
              /usr/include/X11/Xresource.h \
              /usr/include/X11/Xutil.h \
              /usr/include/X11/copyright.h \
              /usr/include/ctype.h \
              /usr/include/errno.h \
              /usr/include/fcntl.h \
              /usr/include/signal.h \
              /usr/include/stdio.h \
              /usr/include/string.h \
              /usr/include/strings.h \
              /usr/include/sys/errno.h \
              /usr/include/sys/fcntl.h \
              /usr/include/sys/file.h \
              /usr/include/sys/stat.h \
              /usr/include/sys/sysmacros.h \
              /usr/include/sys/time.h \
              /usr/include/sys/types.h \
              /usr/include/time.h \
              MenuBox/Menu.h \
              MenuBox/MenuBox.h \
              MenuBox/MenuShell.h

HDRS        = defaults.h \
              general.h \
              interface.h \
              mainmenu.h \
              manager.h \
              menucmd.h \
              processor.h

CFLAGS      = -O

LDFLAGS     =

LIBS        = MenuBox/Menu.o MenuBox/MenuBox.o MenuBox/MenuShell.o \
              /usr/lib/X11/libXaw.a /usr/lib/X11/libXmu.a \
              /usr/lib/X11/libXt.a /usr/lib/X11/libX11.a

LINKER      = cc

MAKEFILE    = Makefile

OBJS        = general.o \
              handler.o \
              interface.o \
              main.o \
              manager.o \
              misc.o

PRINT       = lpsR

PROGRAM     = xnusim

SRCS        = general.c \
              handler.c \
              interface.c \
              main.c \
              manager.c \
              misc.c

all:            $(PROGRAM)

$(PROGRAM):     $(OBJS) $(LIBS)
                @echo -n "Loading $(PROGRAM) ... "
                @/bin/rm -f $(PROGRAM)
                @$(LINKER) $(LDFLAGS) $(OBJS) $(LIBS) -o $(PROGRAM)
                @/bin/chmod 771 $(PROGRAM)
                @echo "done"

clean::         @/bin/rm -f $(OBJS) *~

depend::        @mkmf -f $(MAKEFILE) PROGRAM=$(PROGRAM) DEST=$(DEST)

index::         @etags -wx $(HDRS) $(SRCS)

install:        $(PROGRAM)
                @echo Installing $(PROGRAM) in $(DEST)
                @install -s $(PROGRAM) $(DEST)

print::         @$(PRINT) $(HDRS) $(SRCS)

program:    $(PROGRAM)

tags:       $(HDRS) $(SRCS); @etags $(HDRS) $(SRCS)

update:         $(DEST)/$(PROGRAM)

$(DEST)/$(PROGRAM): $(SRCS) $(LIBS) $(HDRS) $(EXTHDRS)
                @make -f $(MAKEFILE) DEST=$(DEST) install

###
general.o: /usr/include/stdio.h
handler.o: /usr/include/stdio.h /usr/include/strings.h \
        /usr/include/X11/Xlib.h /usr/include/sys/types.h \
        /usr/include/sys/sysmacros.h /usr/include/X11/X.h \
        /usr/include/X11/Xatom.h /usr/include/X11/Intrinsic.h \
        /usr/include/X11/Xutil.h /usr/include/X11/Xresource.h \
        /usr/include/X11/Xos.h /usr/include/string.h /usr/include/fcntl.h \
        /usr/include/sys/file.h /usr/include/sys/fcntl.h /usr/include/time.h \
        /usr/include/sys/time.h /usr/include/X11/Core.h \
        /usr/include/X11/Composite.h /usr/include/X11/Constraint.h \
        /usr/include/X11/StringDefs.h /usr/include/X11/Box.h \
        /usr/include/X11/Command.h /usr/include/X11/Label.h \
        /usr/include/X11/Simple.h /usr/include/X11/copyright.h \
        /usr/include/X11/Xmu.h /usr/include/X11/Dialog.h \
        /usr/include/X11/Form.h /usr/include/X11/Constraint.h \
```

```
        /usr/include/X11/Load.h /usr/include/X11/Scroll.h \
        /usr/include/X11/AsciiText.h /usr/include/X11/Text.h \
        /usr/include/X11/VPaned.h /usr/include/X11/Viewport.h \
        /usr/include/X11/Cardinals.h /usr/include/X11/Shell.h \
        MenuBox/MenuBox.h MenuBox/MenuShell.h MenuBox/Menu.h defaults.h \
        general.h interface.h
interface.o: /usr/include/stdio.h /usr/include/string.h \
        /usr/include/ctype.h /usr/include/signal.h /usr/include/errno.h \
        /usr/include/sys/errno.h /usr/include/X11/Xlib.h \
        /usr/include/sys/types.h /usr/include/sys/sysmacros.h \
        /usr/include/X11/X.h /usr/include/X11/Xatom.h \
        /usr/include/X11/Intrinsic.h /usr/include/X11/Xutil.h \
        /usr/include/X11/Xresource.h /usr/include/X11/Xos.h \
        /usr/include/strings.h /usr/include/fcntl.h /usr/include/sys/file.h \
        /usr/include/sys/fcntl.h /usr/include/time.h /usr/include/sys/time.h \
        /usr/include/X11/Core.h /usr/include/X11/Composite.h \
        /usr/include/X11/Constraint.h /usr/include/X11/StringDefs.h \
        /usr/include/X11/Box.h /usr/include/X11/Command.h \
        /usr/include/X11/Label.h /usr/include/X11/Simple.h \
        /usr/include/X11/copyright.h /usr/include/X11/Xmu.h \
        /usr/include/X11/Dialog.h /usr/include/X11/Form.h \
        /usr/include/X11/Constraint.h /usr/include/X11/Load.h \
        /usr/include/X11/Scroll.h /usr/include/X11/AsciiText.h \
        /usr/include/X11/Text.h /usr/include/X11/VPaned.h \
        /usr/include/X11/Viewport.h /usr/include/X11/Cardinals.h defaults.h \
        general.h interface.h
main.o: /usr/include/X11/Xlib.h /usr/include/sys/types.h \
        /usr/include/sys/sysmacros.h /usr/include/X11/X.h \
        /usr/include/sys/file.h /usr/include/sys/fcntl.h \
        /usr/include/sys/stat.h /usr/include/sys/time.h /usr/include/time.h \
        /usr/include/stdio.h /usr/include/strings.h /usr/include/signal.h \
        /usr/include/errno.h /usr/include/sys/errno.h defaults.h general.h
manager.o: /usr/include/stdio.h /usr/include/strings.h \
        /usr/include/signal.h /usr/include/errno.h /usr/include/sys/errno.h \
        /usr/include/X11/Xlib.h /usr/include/sys/types.h \
        /usr/include/sys/sysmacros.h /usr/include/X11/X.h \
        /usr/include/X11/Xatom.h /usr/include/X11/Intrinsic.h \
        /usr/include/X11/Xutil.h /usr/include/X11/Xresource.h \
        /usr/include/X11/Xos.h /usr/include/string.h /usr/include/fcntl.h \
        /usr/include/sys/file.h /usr/include/sys/fcntl.h /usr/include/time.h \
        /usr/include/sys/time.h /usr/include/X11/Core.h \
        /usr/include/X11/Composite.h /usr/include/X11/Constraint.h \
        /usr/include/X11/StringDefs.h /usr/include/X11/Box.h \
        /usr/include/X11/Command.h /usr/include/X11/Label.h \
        /usr/include/X11/Simple.h /usr/include/X11/copyright.h \
        /usr/include/X11/Xmu.h /usr/include/X11/Dialog.h \
        /usr/include/X11/Form.h /usr/include/X11/Constraint.h \
        /usr/include/X11/Load.h /usr/include/X11/Scroll.h \
        /usr/include/X11/AsciiText.h /usr/include/X11/Text.h \
        /usr/include/X11/VPaned.h /usr/include/X11/Viewport.h \
        /usr/include/X11/Cardinals.h MenuBox/MenuBox.h MenuBox/MenuShell.h \
        /usr/include/X11/Shell.h MenuBox/Menu.h defaults.h general.h \
        interface.h menucmd.h manager.h mainmenu.h
misc.o: /usr/include/stdio.h /usr/include/strings.h /usr/include/X11/Xlib.h \
        /usr/include/sys/types.h /usr/include/sys/sysmacros.h \
        /usr/include/X11/X.h /usr/include/X11/Xatom.h \
        /usr/include/X11/Intrinsic.h /usr/include/X11/Xutil.h \
        /usr/include/X11/Xresource.h /usr/include/X11/Xos.h \
        /usr/include/string.h /usr/include/fcntl.h /usr/include/sys/file.h \
        /usr/include/sys/fcntl.h /usr/include/time.h /usr/include/sys/time.h \
        /usr/include/X11/Core.h /usr/include/X11/Composite.h \
        /usr/include/X11/Constraint.h /usr/include/X11/StringDefs.h \
        /usr/include/X11/Box.h /usr/include/X11/Command.h \
        /usr/include/X11/Label.h /usr/include/X11/Simple.h \
        /usr/include/X11/copyright.h /usr/include/X11/Xmu.h \
        /usr/include/X11/Dialog.h /usr/include/X11/Form.h \
        /usr/include/X11/Constraint.h /usr/include/X11/Load.h \
        /usr/include/X11/Scroll.h /usr/include/X11/AsciiText.h \
        /usr/include/X11/Text.h /usr/include/X11/VPaned.h \
        /usr/include/X11/Viewport.h /usr/include/X11/Cardinals.h \
        /usr/include/X11/Shell.h defaults.h interface.h processor.h
```

defaults.h, page 1

```
/* the defaults of Mux are defined in this header file, includes Simulator name, and
   maximum sizes etc */

#define MAXTEXT 256
#define MAXARG 30
#define MAXCHAR 1024
#define MAXHISTORY (MAXTEXT * MAXCHAR)
#define MAXDIALLEN 80
#define PROMPTFONT 480
#define MAXWORDSIZE 8
#define MAXBREAKS 30

#define SimulatorName " NuSim Simulator "

#define Simulator "/hprg/NuSim/nusim"

#define SHELL "/bin/csh"
#define SHELLPROG "recur.csh"

#define DEFAULT_TITLE_FONT "-adobe-courier-bold-o-normal--18-180-75-75-m-110-iso8859-1"
#define DEFAULT_SUBTTL_FONT "-adobe-courier-bold-r-normal--14-140-75-75-m-90-iso8859-1"

#define ERRORCRY "%%%ERROR   : "
```

```
/*
 * Copyright (c) 1989 Regents of the University of California
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that this notice is preserved and that due credit is given
 * to the University of California at Berkeley. The name of the University
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission. This software
 * is provided "as is" without express or implied warranty.
 */

#ifndef lint
static char sccsid[] = "@(#)general.c     3.1   6/26/89";
#endif /* not lint */

/* general.c:  some basic functions useful for most programs */

#include <stdio.h>

error( str, type )
    char *str;
    int type;
{
    fprintf(stderr, "%s\n", str);
    if (type < 2) exit(type);
}

inchr(str, c)
    char *str, c;
{
    int i;
    for(i=0; *str != '\0' && *str != c; str++, i++);
    if ( *str != c ) return -1;
    return i;
}

instr(s1, s2)
    char *s1, *s2;
{
    int i, j;
    for(i=0; *s1 != '\0'; s1++, i++) {
            for(j=0; s1[j] != '\0' && s2[j] != '\0' && s1[j] == s2[j]; j++);
            if (s2[j] == '\0') return i;
            if (s1[j] == '\0') return -1;
    }
    return(-1);
}

hextoi(str)
    char *str;
{
    int i, j;

    for(i=0; *str != '\0' &&
                (j = inchr("0123456789abcdefABCDEF", *str)) != -1; str++) {
            if (j > 15 ) j -= 6;
            i = i*16 + j;
    }
    return i;
}

char *itohex(val, size)
    int val, size;
{
    int i;
    char *p;

    p = (char *) calloc(size+1, sizeof(char));
    p[size]= '\0';

    for(i = size-1; i >= 0 && val >= 1; i--) {
            p[i] = "0123456789ABCDEF"[val%16];
            val = val >> 4;
    }
    for(; i >= 0; i--) p[i] = '  ';
    return p;
}
```

general.h, page 1

```
/* Some basic definition useful for several programs */
#define CALLOC(n, t) ( (t *) calloc((n), sizeof(t)))
#define MALLOC( t ) CALLOC(1, t)
#define LARGE (0x7fffffff)

#define forever for(;;)
#define min(x, y) (((x) < (y))?(x):(y))
#define max(x, y) (((x) > (y))?(x):(y))

extern char *itohex();
```

```
/*
 * Copyright (c) 1989 Regents of the University of California
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that this notice is preserved and that due credit is given
 * to the University of California at Berkeley. The name of the University
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission. This software
 * is provided "as is" without express or implied warranty.
 */

#ifndef lint
static char sccsid[] = "@(#)handler.c     4.1  9/11/89";
#endif /* not lint */

/* handler.c : module that talks between MULTIX and nusim
 */

#include <stdio.h>
#include <strings.h>
#include <ctype.h>
#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Box.h>
#include <X11/Command.h>
#include <X11/Dialog.h>
#include <X11/Label.h>
#include <X11/Load.h>
#include <X11/Scroll.h>
#include <X11/AsciiText.h>
#include <X11/VPaned.h>
#include <X11/Viewport.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>

#include "MenuBox/MenuBox.h"
#include "MenuBox/MenuShell.h"
#include "MenuBox/Menu.h"
#include "general.h"
#include "defaults.h"
#include "interface.h"

/* basic set up variables for handler */
static int runcount, stepcount; /* the no of times to execute Run and Step */
static int breakptcount, breakpt[MAXBREAKS][2];
static int breaktm; /* initial no breaktime, set to "infinity" */
Widget top;

/* initializes this module */
void handler_init(pass)
    Widget pass;
{
    runcount=0;
    stepcount=1;
    if (pass != NULL) top = pass;
    breaktm = LARGE;
}

/* display message on the "help window" part of the display */
void help(widget, text, event)
    Widget widget;
    char  *text;
    XCrossingEvent *event;
{
    extern Widget whelp;
    Arg help_arg;

    XtSetArg(help_arg, XtNlabel, text);
    XtSetValues(whelp, &help_arg, 1);
}

/* exit from program */
void quit(widget, text, event)
    Widget widget;
    char *text;
    XCrossingEvent *event;
{
    extern int killChild();
    extern void killWindows();
    char *tmp;

/*    tmp = dialog( "Really (Y/N) ?" );
    if (tmp[0] != 'Y' && tmp[0] != 'y') return;
*/
    sendMsg(COMMAND, "quit\n", 0);
    killChild();
    killWindows();
    exit(0);
}

/* reset */
void reset(widget, text, event)
    Widget widget;
    char *text;
    XCrossingEvent *event;
{
    extern int killChild();

    sendMsg(RESET, NULL, 1);
    resetmanager();
}

/* perform the "load" operation */
void load(widget, client, call)
    Widget widget;
    caddr_t client, call;
{
    sendMsg(LOAD, NULL, 1);
}

/* step as many steps as are necessary, in response to the "step" command */
void step(widget, client, call)
    Widget widget;
    caddr_t client, call;
{
    int i;

    sendMsg(STEP, NULL, stepcount);
}

/* performs the RUN, also takes the configured number of runs */
void run(widget, client, call)
    Widget widget;
    caddr_t client, call;
{
    if (runcount == 0)
            sendMsg(RUN, NULL, 0);
    else
            sendMsg(RUN, NULL, runcount);
```

```
/* attempts to get the Proc/Task environment under which the break is suppose 2 function */
void breakenv(widget, client, call)
    Widget widget;
    caddr_t client, call;
{
    static int working = 0;
    int envtask, envprocess, i;
    char sender[40];
    Widget wdial, wbox, wsend[2], wresp[2];
    XEvent event;
    Arg arg[MAXARG];
    Cardinal argn;
    char reply[2][MAXCHAR], send[2][MAXCHAR];

    if (working == 1) return; /* semaphore */

    working = 1;
    updateenv(&envtask, &envprocess);
    argn = 0;
    XtSetArg(arg[argn], XtNborderWidth, 2); argn++;

    /* creates the pop up shell for this function */
    wdial = XtCreatePopupShell("EnvShell", shellWidgetClass, top, arg, argn);

    wbox = XtCreateManagedWidget("box", boxWidgetClass, wdial, NULL, 0);

    XtAddEventHandler(wbox, EnterWindowMask, 0, help, (caddr_t) "Hit return when done");

    /* for each of Processor/task, print current environment and request new one */
    sprintf(send[0], "Proc Env (%3d ) :", envprocess);
    sprintf(send[1], "Task Env (%3d ) :", envtask);

    for(i=0; i<2; i++) {
        argn = 0;
        XtSetArg(arg[argn], XtNlength, MAXDIALLEN); argn++;
        XtSetArg(arg[argn], XtNstring, send[i]); argn++;
        XtSetArg(arg[argn], XtNborderWidth, 0); argn++;
        XtSetArg(arg[argn], XtNeditType, XttextRead); argn++;
        XtSetArg(arg[argn], XtNwidth, PROMPTFONT); argn++;
        XtSetArg(arg[argn], XtNinsertPosition, strlen(send[i])+1); argn++;
        XtSetArg(arg[argn], XtNsensitive, False); argn++;

        wsend[i] = XtCreateManagedWidget((i==0)?"procdisp":"taskdisp",
                                         asciiStringWidgetClass, wbox, arg, argn);

        argn = 0;
        bzero(reply[i], sizeof(char)*4);
        XtSetArg(arg[argn], XtNlength, MAXDIALLEN); argn++;
        XtSetArg(arg[argn], XtNborderWidth, 1); argn++;
        XtSetArg(arg[argn], XtNstring, reply[i]); argn++;
        XtSetArg(arg[argn], XtNeditType, XttextEdit); argn++;
        XtSetArg(arg[argn], XtNwidth, PROMPTFONT); argn++;
        XtSetArg(arg[argn], XtNleftMargin, 2); argn++;
        XtSetArg(arg[argn], XtNinsertPosition, 0); argn++;

        wresp[i] = XtCreateManagedWidget((i==0)?"procrep":"taskrep",
                                         asciiStringWidgetClass, wbox, arg, argn);
    }

    XtPopup( wdial );

    XtRealizeWidget( wdial );

    /* keep getting input until a "return" is hit */
    while ( reply[0][strlen(reply[0])-1] != '\n' &&
            reply[1][strlen(reply[1])-1] != '\n' ) {
        XtNextEvent(&event);
        XtDispatchEvent(&event);
    }

    /* destroy dialog popup widget */
    XtDestroyWidget( wdial );

    while( XtPending() ) {
        XtNextEvent(&event);
        XtDispatchEvent(&event);
    }

    /* set up the call for update */
    if (reply[0][0] != '\0' && reply[0][0] != '\n')
        envprocess = atoi(reply[0]);
    if (reply[1][0] != '\0' && reply[1][0] != '\n')
        envtask = atoi(reply[1]);
    sprintf(reply[0], "Env: Proc (%3d ) and Task (%3d )", envprocess, envtask);
    sprintf(reply[1], "%d %d", envprocess, envtask);
    sendMsg( BREAKENV, reply[1], 0 );

    help(widget, reply[0], (caddr_t) NULL);
    working = 0;
}


/* is called when one of the button for the breakpopup is pressed,
   deduces which is the right button pressed and performs the button request */
dispbreakpt(widget, j, call)
    Widget widget;
    caddr_t j, call;
{
    int testbit, i, call_type;
    char *tmp;

    call_type = (int) j;
    switch(call_type) {
    case -1: /* -1 is defined as deleting, so call dobreak, and update breakpt list */
        dobreak(0, -1);
        break;
    case -2: /* -2 is when user wants to input his own address, so get input */
    case -4:
        tmp = dialog( "Address to use:" ); /* get input */
        testbit = 1;
        for(i=0; i < strlen(tmp) && isspace(tmp[i]); i++);
        if (tmp[i] == '0' && (tmp[i+1] == 'x' || tmp[i+1] == 'X')) /* test for hexa */
            testbit = hextoi(tmp+2);
        else { /* generic assumed */
            for(i=0; i < strlen(tmp) && testbit != -1; i++)
                if (!isspace(tmp[i]) && !isdigit(tmp[i])) {
                    if (isxdigit(tmp[i])) testbit = 2;
                    else testbit = -1;
                }
            if (testbit == -1)
                help(widget, "only Numbers allowed (hex/dec)", (caddr_t) NULL);
            else if (testbit == 2)
                testbit = hextoi(tmp);
            else testbit = atoi(tmp);
        }
        if (testbit < 0) return; /* illegal */
        dobreak(testbit, (call_type==-2)?3:4);
        break;
    case -3: /* -3 is current cursor position in list window */
    case -5:
```

```
            testbit = getlistposn(); /* get cursor position in list window */
            dobreak(testbit, (call_type==-3)?1:2); /* update and call */
            break;
    default:
            /* delete specific element of the break list, "call_type" is which elemen */
            dobreak(breakpt[call_type][0], 0); /* delete that element */
    }
}

/* handles the breakpoint calls */
void breakpoint(widget, client, call)
    Widget widget;
    caddr_t client, call;
{
    PopupMenu *menu = NULL;
    int i;

    updatebreakpt(breakpt, &breakptcount);
    menu = MenuCreate( top, widget, "==Breakpoints==" );
    if (breakptcount < MAXBREAKS) { /* set break point calls if not overloaded */
            MenuAddSelection(menu, "Cursor break posn", "Click to set", dispbreakpt, help, -3);
            MenuAddSelection(menu, "Cursor trace posn", "Click to set", dispbreakpt, help, -5);
            MenuAddSelection(menu, "Type breakpt", "Click for dialog", dispbreakpt, help, -2);
            MenuAddSelection(menu, "Type tracept", "Click for dialog", dispbreakpt, help, -4);
    }
    if (breakptcount > 1) /* delete all useful only when more than 1 breakpt set */
            MenuAddSelection(menu, "Delete ALL", "Click to Delete all breakpoints",
                             dispbreakpt, help, -1);
    for( i=0; i < breakptcount; i++) { /* display each of the breakpoints for click del */
            char tmp[80], *hexa;
            hexa = (char *) itohex(breakpt[i][0], 6);
            sprintf(tmp, "Delete 0x%s (%c)", hexa,
                    (breakpt[i][1] == 1?'t':'b')); /* which to delete */
            MenuAddSelection(menu, tmp, "Click to delete", dispbreakpt, help, i);
    }
    MenuReady( menu );
    XtPopup(menu->shell);
}

/* button handling, display break time */
void dispbreaktm(widget, i, call)
    Widget widget;
    caddr_t i, call;
{
    char tmp[80], *p;

    if ((int) i == -1) {
            sprintf(tmp, "%d", LARGE);
            sendMsg(BREAKTM, tmp, 0);
    } else {
            sprintf(tmp, "Set breaktime value (cur: %d)", breaktm);
            p = dialog( tmp );

            if (atoi(p) > 0) {
               sendMsg(BREAKTM, p, 0);
            }
    }
}

/* set or delete break time */
void breaktime(widget, client, call)
    Widget widget;
    caddr_t client, call;
{
    PopupMenu *menu = NULL;

    updatebreaktm(&breaktm);
    menu = MenuCreate( top, widget, "==Breaktime==" );
    if (breaktm != LARGE) {
            char *tmp, str[80];
            tmp = itohex(breaktm, 6);
            sprintf(str, "Remove breaktime: 0x %s", tmp);
            MenuAddSelection(menu, str, "Click to select", dispbreaktm, help, -1);
    }
    MenuAddSelection(menu, "set/change Breaktime", "Click to select",
                     dispbreaktm, help, 1);
    MenuReady( menu );
    XtPopup(menu->shell);
}

void displaytask(widget, i, call)
    Widget widget;
    caddr_t i, call;
{
    (void) manageTask(i, top);
}

void taskMain(widget, client, call)
    Widget widget;
    caddr_t client, call;
{
    static PopupMenu *menu = NULL;

    if (menu == NULL) {
            int i;

            menu = MenuCreate( top, widget, "TaskTable" );
            MenuBind(widget, "TaskTable", "<BtnUp>");
            for(i=0; i < NUMPROC; i++) {
               char tmp[80];
               sprintf(tmp, "TTE(%d)", i);
               MenuAddSelection(menu, tmp, "Click to choose (on/off)", displaytask, help, i);
            }
            MenuReady( menu );
            XtPopup(menu->shell);
    }
}


void displayprocess(widget, i, call)
    Widget widget;
    caddr_t i, call;
{
    (void) manageProc(i, top);
}

void procMain(widget, client, call)
    Widget widget;
    caddr_t client, call;
{
    static PopupMenu *menu = NULL;

    if (menu == NULL) {
            int i;

            menu = MenuCreate( top, widget, "PROCESSOR" );
            MenuBind(widget, "PROCESSOR", "<BtnUp>");
            for(i=0; i < NUMPROC; i++) {
               char tmp[80];
               sprintf(tmp, "Processor(%d)", i);
               MenuAddSelection(menu, tmp, "Click to choose", displayprocess, help, i);
            }
```

```
        MenuReady( menu );
        XtPopup(menu->shell);
    }


/* handle dialog controls */
char *dialog( str )
    char *str;
{
    static int working=0;
    static char receiver[MAXDIALLEN], sender[MAXDIALLEN];
    static Widget wdial, wbox, wsend, wresp;
    XEvent event;
    Arg arg[MAXARG];
    Cardinal argn;
    char *reply;

    argn = 0;
    sprintf(sender, "%dx%d", 40, PROMPTFONT);
    XtSetArg(arg[argn], XtNgeometry, sender); argn++;
    XtSetArg(arg[argn], XtNborderWidth, 2); argn++;

    wdial = XtCreatePopupShell("DialogShell", shellWidgetClass, top, arg, argn);

    wbox = XtCreateManagedWidget("box", boxWidgetClass, wdial, NULL, 0);

    XtAddEventHandler(wbox, EnterWindowMask, 0, help, (caddr_t)
                "Dialog Box: Hit return when done");

    argn = 0;
    strcpy(sender, str);
    XtSetArg(arg[argn], XtNlength, MAXDIALLEN); argn++;
    XtSetArg(arg[argn], XtNstring, sender); argn++;
    XtSetArg(arg[argn], XtNborderWidth, 0); argn++;
    XtSetArg(arg[argn], XtNeditType, XttextRead); argn++;
    XtSetArg(arg[argn], XtNwidth, PROMPTFONT); argn++;
    XtSetArg(arg[argn], XtNinsertPosition, strlen(sender)+1); argn++;
    XtSetArg(arg[argn], XtNsensitive, False); argn++;

    wsend = XtCreateManagedWidget("sender", asciiStringWidgetClass, wbox, arg, argn);

    argn = 0;
    bzero(receiver, sizeof(char)*MAXDIALLEN);
    XtSetArg(arg[argn], XtNlength, MAXDIALLEN); argn++;
    XtSetArg(arg[argn], XtNborderWidth, 1); argn++;
    XtSetArg(arg[argn], XtNstring, receiver); argn++;
    XtSetArg(arg[argn], XtNeditType, XttextEdit); argn++;
    XtSetArg(arg[argn], XtNwidth, PROMPTFONT); argn++;
    XtSetArg(arg[argn], XtNleftMargin, 2); argn++;
    XtSetArg(arg[argn], XtNinsertPosition, 0); argn++;

    wresp = XtCreateManagedWidget("response", asciiStringWidgetClass, wbox, arg, argn);

    XtPopup( wdial );

    XtRealizeWidget( wdial );

    while ( receiver[strlen(receiver)-1] != '\n' ) {
        XtNextEvent(&event);
        XtDispatchEvent(&event);
    }

    XtDestroyWidget( wdial );

    while( XtPending() ) {
        XtNextEvent(&event);
        XtDispatchEvent(&event);
    }

    receiver[strlen(receiver)-1] = '\0';
    reply = CALLOC( strlen(receiver) + 1,char);
    strcpy(reply, receiver);
    return( reply );
}

/* CONFIGURATION SUBMODULE */

void buttons(widget, client, call)
    Widget widget;
    caddr_t client, call;
{
    extern void configTask();
    char tmp[80];

#ifdef DEBUG
    sprintf(tmp, "Called buttons: (%d)", (int) client);
    error(tmp, 5);
#endif
    if ((int) client == 2) configTask(top);
    if ((int) client == 3) configProc(top);
}

void control(widget, client, call)
    Widget widget;
    caddr_t client, call;
{
    static int working = 0; /* semaphore */
    char tmp[MAXCHAR], *reply;
    int tmpval;

    if (working == 1) return;
    working = 1;
    if ((int) client == 0) {
        sprintf(tmp, "New stepsize: (current = %d)", stepcount);
    } else {
        sprintf(tmp, "New run mode [n*%d steps]: (cur = %d)", RUNSIZE, runcount);
    }
    reply = dialog( tmp );
    if (reply[0] == '\0' || reply[0] == '\n') {
        help(widget, "No change in value", 0);
    } else {
        tmpval = atoi(reply);
        sprintf(tmp, "New val: %d", tmpval);
        help(widget, tmp, 0);
    }
    if ((int) client == 0)
        stepcount = tmpval;
    else
        runcount = tmpval;
    working = 0;
}


void config(widget, client, call)
    Widget widget;
    caddr_t client, call;
{
    static PopupMenu *configw = NULL;

    if (configw == NULL) {
        configw = MenuCreate( top, widget, "CONFIGURATION" );
```

```
        MenuBind(widget, "CONFIGURATION", "<BtnUp>");
        MenuAddSelection(configw, "Step", "Set step size", control, help, 0);
        MenuAddSelection(configw, "Run", "Set run size/mode", control, help, 1);
        MenuAddSelection(configw, "Tasks", "Select Task entries", buttons, help, 2);
        MenuAddSelection(configw, "Processors", "Select Proc entries", buttons, help,3);
/*      MenuAddSelection(configw, "Summary", "Select Summary entries", buttons, help, 4); */
        MenuReady(configw);
        XtPopup(configw->shell);
    }

void killWindows()
{


void summMain(widget, client, call)
    Widget widget;
    caddr_t client, call;
{
    error("summary: unimplemented", 3);
}
```

```
/*
 * Copyright (c) 1989 Regents of the University of California
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that this notice is preserved and that due credit is given
 * to the University of California at Berkeley. The name of the University
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission. This software
 * is provided "as is" without express or implied warranty.
 */

#ifndef lint
static char sccsid[] = "@(#)interface.c   4.1   9/11/89";
#endif /* not lint */

/* interface.c : module that talks between xnusim and nusim
 */

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <signal.h>
#include <errno.h>
#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Box.h>
#include <X11/Command.h>
#include <X11/Dialog.h>
#include <X11/Label.h>
#include <X11/Load.h>
#include <X11/Scroll.h>
#include <X11/AsciiText.h>
#include <X11/VPaned.h>
#include <X11/Viewport.h>
#include <X11/Cardinals.h>

#include "defaults.h"
#include "general.h"
#include "interface.h"

extern void reposition();
int processor[NUMPROC], task[NUMTASK];
int linecount;
int *line, *linestart;
int maxline;
int lastpos=0;
int more; /* tells xnusim to expect more output from nusim */

XEvent event;
extern char *list;
static Widget mainscreen, subscreen, helpscreen;
static int finalcount, runned, position, loaded;
static char *loadfile;
static char *filename;
int Finalpos;

/* initializes this module: called once.  set up size of lines
   setup == 0 means First call, have to set up listing array.
   setup != 0 means only need to reset other things
 */
void init_interface(size, setup)
    int size, setup;
{
  more = 1;
  loaded = 0;
  finalcount = 0;
  runned = 0;
  Finalpos = 0;
  filename = NULL;
  linecount = 0;
  position = 0;
  if (setup == 0) {
        maxline = max(1, size) * (MAXHISTORY / 40);
        {
          char temp[80];
          sprintf(temp, "Maxline: %d (%d) \n", maxline, size);
          error(temp, 3);
        }
        line = CALLOC(maxline, int);
        linestart = CALLOC(maxline, int);
  }
}


/* needline:
   returns the next line read from the buffer.
   if type == 0; needline treats it as a first read request.
   returns NULL when prompt line is encountered. */
char *needline(type)
   int type;
{
   static int none;
   static char buf[MAXCHAR];
   static char line[MAXCHAR];
   static int i, j, k;

   if (type == 0) {
        i = 0;
        buf[0] = '\0';
        none = 0;
   }
   while ((j=inchr(buf, '\n')) == -1) { /* if no newline, try reading again */
        if (none != 0) return NULL; /* prompt was at previous read. */
        i = strlen(buf); /* read line without destroying the leftover in buf */
        if ((i+=MessageRead( buf+i, MAXCHAR-i )) < 0 ) {
          perror("reading from simulator at needline");
          exit(1);
        }
        for(k=i-1; k > 0 && isspace(buf[k]); k--);
        if (buf[k] == PROMPT) none = 1; /* if promptline encountered, stop */
   }
   strncpy(line,buf,j+1); /* return 1 line via 'line' */
   for(k=j+1; k < i+1; k++) buf[k-j-1] = buf[k]; /* shorten buf */
   i=k-j-1;
   line[j+1] = '\0';
   for(; j > 0 && isspace(line[k]); j--);
   if (line[j] == PROMPT) return NULL; /* don't return a line with prompt */
   return line;
}

/* doload:
   attempts to find out what file the user wants loaded and loads it */
static void doload()
{
   int i;
   char temps[MAXCHAR];
   FILE *fd;

   if (loaded != 0) return;
```

```
    linecount = 0;
    finalcount = 0;

    if (filename == NULL)
            filename = dialog("Filename :"); /* ask for a filename */
    if ((fd = fopen(filename, "r")) == (FILE *) NULL) {
            sprintf(temps, "%s  File %s cannot be opened (fopen)",
                    ERRORCRY, filename);
            help(helpscreen, temps, (caddr_t) NULL);
            return;
    }
    close(fd);

    /* LOAD */
    strcpy(temps, "load ");
    strcat(temps, filename);
    strcat(temps, "\n");
    if (MessageWrite( temps, 1 ) < 0) {
            perror("Writing Load");
            error(temps, 1);
    }
    more++;
}

/* gethex:
   Pre: s is a double pointer to a string of the form " 0x <hex_addr> "
   Post: returns the integer value of the <hex_addr> */
int gethex(s)
    char **s;
{
    int i;

    for(; **s != '\0' && isspace(**s); (*s)++);
    if ((*s)[0] == '0' && ((*s)[1] == 'x' || (*s)[1] == 'X')) (*s) += 2;
    for(; **s != '\0' && isspace(**s); (*s)++);
    i = hextoi(*s);
    for(; **s != '\0' && inchr("0123456789abcdefABCDEF", **s) != -1; (*s)++);
    return i;
}

/* findplace: finds the correct place in the list window where the string is located */
int findplace(str, posn)
{
    return posn + instr(loadfile+posn, str);
}


/* loadprocess:
   Pre:  s is assumed to be of the form
       (rubbish1"filename"1"rubbish1[" "start_addr" "-" "end_addr" "]]*
       and the program loaded in simulator.
   Post: progstart, progend and the line[] array will have the
       programstart address, programend address and
           line[] will have the memory addresses of that line */
void loadprocess(buf)
    char *buf;
{
    char *string, *s, tmp[MAXCHAR], *filename;
    int i, j, addr, index, again;
    int progstart, progend, progblk;
    FILE *fd;

    loaded = 1;

    s = CALLOC(strlen(buf)+1, char);
    strcpy(s, buf);

#ifdef DEBUG
    error("Load processing", 3);
    error(s, 3);
#endif
    if (instr(s, "load") == -1) return;

    if ((i=inchr(s, '\'')) == -1) {
            error("No filename in string", 3);
            return;
    }
    s+=(i+1);
    if ((i=inchr(s, '\'')) == -1) {
            error("Parsing string for quotes", 3);
            return;
    }
    s[i] = '\0';
    filename = CALLOC(strlen(s)+1, char);
    strcpy(filename, s);

    /* read out the start-end addresses pair as stated in "[start-end]"
       where each address is of the form " 0x hex_value " */
    for(s+=(i+1); *s != '\0' && *s != '['; s++);
    s++;

    progstart = gethex(&s);
    for(; *s != '\0' && isspace(*s); s++);
    s++;
    progend = gethex(&s);

    loadprocess(s);
#ifdef DEBUG
    sprintf(tmp, "File: %s, [%d - %d]", filename, progstart, progend);
    error(tmp, 3);
#endif

    if ((fd = fopen(filename, "r")) == (FILE *) NULL) {
            sprintf(tmp, "%s  File %s cannot be opened (fopen)",
                    ERRORCRY, filename);
            help(helpscreen, tmp, (caddr_t) NULL);
            return;
    }
    fprintf(stderr, "Loading: %s\n", filename);
    index = linecount;

    /* get where each line begin and putting it into listbuffer */
    for(; (linecount < maxline) &&
            (fgets( tmp+12, MAXCHAR, fd) != NULL); linecount++ ) {
            for(i=0; i < 10; i++) tmp[i] = ' '; /* leave space for line numbering */
            tmp[i++] = ':';
            tmp[i++] = ' ';
            for(; tmp[i] != '\0' && isspace(tmp[i]); i++);
            if (tmp[i] != '\0' && tmp[i] != '%') finalcount=linecount;
            linestart[linecount] = putList2( tmp, 0 );
            if (linecount != 0 && linecount % 500 == 0)
              fprintf(stderr, "Loaded: %d lines\n", linecount);
            sprintf(tmp, "Line: %d at %d (%d)", linecount, linestart[linecount],
                    finalcount);
#ifdef DEBUG
            error(tmp, 3);
#endif
    }
    linecount=finalcount;
    fprintf(stderr, "%d\n", linecount);
    if (linecount==0) {
            sprintf(tmp, "%s File %s empty\n", ERRORCRY, filename);
            help(helpscreen, tmp, (caddr_t) NULL);
```

```
        return;
    }
    close(fd);

    for(i=index; i < linecount; i++) line[i] = -1;
    j = 0;

    /* now that we have a knowledge of where the program lies in memory
       we make the simulator print out the codes as it preceived it in
       terms of memory addresses */

    for(progblk = progstart; progblk < progend; progblk+=100) {

        sprintf(tmp, "code %d %d\n", progblk, min(progblk+99, progend));
/*      error(tmp, 3); */
        fprintf(stderr, " . "); fflush(stderr);
        again = 0;
        if (MessageWrite( tmp, 0 ) < 0) {
          perror("Writing code print");
          error("Writing code", 1);
        }

        while ((string = needline(again++)) != NULL ) {
          while (string == NULL || (string[0] == '@' && string[1] == 'E')) {
                if (string != NULL) progblk++;
                sprintf(tmp, "code %d %d\n",
                        progblk, min(progblk+99, progend));
                if (MessageWrite( tmp, 0 ) < 0) {
                  perror("Writing code print");
                  error("Writing code", 1);
                }
                string = needline(0);
                fprintf(stderr, "?"); fflush(stderr);
/*              fprintf(stderr, "\nCode: %s\n Line: %s", tmp, string); */
          }
          if ((++j % 500) == 0) fprintf(stderr, "Processed: %d lines\n", j);

          /* Parse " 0x addr_hex : command " recursively */
          addr = gethex(&string); /* " 0x addr_hex " */
          for(; *string != '\0' && isspace(*string); string++);

          if (*string != '\0') string++; /* : */
          for(; *string != '\0' && isspace(*string); string++);
          for(i=0; string[i] != '\0' && !isspace(string[i]); i++); /* comm */
          if ( i != 0 ) {                   /* search for command */
                string[i] = '\0';
                position = findplace( string, position+1 ); /* find it's posn */
                for(; index < linecount && position > linestart[index+1];
                  index++);             /* find which "line" it's on */
                if (index < linecount) {
                  line[index] = addr; /* insert the address into line */
                  if (addr > 0) {  /* updates the loadfile image with linenumber */
                        int where = position-1;
                        int linei;
                        char *hexaddr;

                        for(; where >=0 && loadfile[where] != '\n'; where--);
                        where++;
                        hexaddr = itohex(addr, 8);
                        loadfile[where] = '0'; loadfile[where+1] = 'x';
                        for(linei=2; linei < 10; linei++)
                          loadfile[linei+where] = *(hexaddr++);
                  }
                } else {
                  char bug[80];
                  sprintf(bug, "Index wrap: %d (%d) %s", index, linecount, tmp);
                  error(bug, 3);
                  index = linecount >> 1;
                }
                /* go to next line */
#ifdef DEBUG
                fprintf(stderr, "(%5d) %5d is %s\n", index, position, string);
#endif
                string += i+1;
                for(i=0; string[i] != '\0' && string[i] != '\n'; i++);
                if (string[i] != '\0') i++;
                string += i;
          }
        }

    }
    fprintf(stderr, "\n"); fflush(stderr);
    putList2( NULL, 1 );
    for(i=0; i < linecount && line[i] == -1; i++);
    if (i < linecount) reposition(i);
    else reposition(0);
}

/* findLine:
   Pre:  position is the current cursor position
   Post: returns which line number (in base 10) of which position belongs. */
int findLine( position )
    int position;
{
    int i;
    for(i=0; i < linecount && linestart[i] <= position; i++);
    return --i;
}

/* called by handler's "breakenv" button to get current env value from stat */
updateenv(task, proc)
    int *task, *proc;
{
    int i;
    char *s;
    if (MessageWrite( "stat\n", 0 ) < 0) {
          perror("Writing stat (task, proc)");
          exit(1);
    }
    /* assumed format: 3 lines. 3rd line's 2nd & 3rd field is proc and task */
    (void) needline(0);
    (void) needline(1);
    s = needline(2);
    i = inchr(s, ' : ');
    i += inchr(s+i+1, ' : ');
    i++;
    *proc = atoi(s+i+1);
    i += inchr(s+i+1, ' : ');
    i++;
    *task = atoi(s+i+1);
}

/* called by handler's "breaktime" button to get current break time value */
updatebreaktm(bt)
    int *bt;
{
    int i;
    char *s;
    if (MessageWrite( "stat\n", 0 ) < 0) {
          perror("Writing stat (task, proc)");
          exit(1);
    }
```

```
    /* assumed format: 3 lines. 3rd line's 1st field is breaktime */
    (void) needline(0);
    (void) needline(1);
    s = needline(2);
    i = inchr(s, ' : ');
    *bt = atoi(s+i+1);
}

/* called by handler's "breakpoint" button to get current breakpoint set */
updatebreakpt(bp, count)
    int bp[][2], *count;
{
    int i;
    char *s;
    if (MessageWrite( "sb\n", 0 ) < 0) {
            perror("Writing sb");
            exit(1);
    }
    s = needline(0);
    *count = 0;
    /* if first line dont have ':' means nusim replied with no breakpt set */
    if (inchr(s, ' : ') == -1) return;
    while ( (s = needline(1)) != NULL ) { /* while there's a line */
            /* assumed format: <count>. [<label>:]<address> (<b/t>) */
            i = inchr(s, ' . '); s+=(i+1);
            if ((i=inchr(s, ' : ')) != -1) s+=(i+1); /* skip "optional" field */
            bp[*count][0] = gethex(&s);
            i = inchr(s, ' ('); s+=(i+1);
            bp[*count][1] = (((*s)==' t')?1:2);
            (*count)++;
    }
}

/* dobreak:
   mode = -1: delete all, 0: delete, odd, even: insert (b or t respectively)
   sends the correct break command to the simulator
 */
int dobreak( linenum, mode )
    int linenum, mode;
{
    int i;
    char tmp[MAXCHAR];

    switch(mode) {
    case 0:
            sprintf(tmp, "rm %d\n", linenum);
            break;
    case -1:
            sprintf(tmp, "rmall\n");
            break;
    case 1:
    case 2:
            i = findLine( linenum )-1;
            if (i < 0) i = 0;
            for(; (linenum = line[i]) < 0; i++);
            sprintf(tmp, "bp %d %c\n", linenum, (mode==1)?'b':'t');
            break;
    case 3:
    case 4:
            sprintf(tmp, "bp %d %c\n", linenum, (mode==3)?'b':'t');
            break;
    default:
            error("Passed unknown value to dobreak", 3);
            return;
    }
    if (MessageWrite( tmp, 1 ) < 0) {
            perror("Writing (dobreak)");
            exit(1);
    }
    reposition(i); /* goto that line where the break was set */
    more++;
    return linenum;
}

/* sendMsg:
   Pre: sendcomm is the predefined set of commands to execute in the
        simulator.
        str is an optional argument string to the command
        times is an optional number of times to execute this command.
   Post: the sendcomm is executed and more updated if necessary */
void sendMsg( sendcomm, str, times )
    int sendcomm, times;
    void *str;
{
    static int i, k, lock=0;
    char tmp[MAXCHAR];

    if (lock == 1) return;
    switch(sendcomm) {
    case LOAD:
            lock=1; /* lock so u cant try to load Two files */
            filename = (char *) str;
            doload();
            lock=0;
            break;
    case STEP:
            for(i=0; i < times-1; i++) {
              if (MessageWrite( "s\n", 0 ) < 0) {
                      perror("Writing Load");
                      error("s\n", 1);
              }
              k=0;
              while(needline(k++) != NULL); /* flush buffer */
            }
            if (MessageWrite( "s\n", 1 ) < 0) { /* echo it this time */
              perror("Writing Load");
              error("s\n", 1);
            }
            more = 1;
            break;
    case RUN:
            if (MessageWrite( (runned == 0)?"run\n":"c\n", 1 ) < 0) {
              perror("Writing Load");
              error("run\n", 1);
            }
            runned = 1;
            more = 1;
            break;
    case PROCESSOR:
            processor[(int) str] = times;
            if (processor[(int) str] != 0)
              (void) updateProc((int) str);
            break;
    case TASK:
            task[(int) str] = times;
            if (task[(int) str] != 0)
              (void) updateTask((int) str);
            break;
    case BREAKENV:
            sprintf(tmp, "be %s\n", (char *) str);
            if (MessageWrite( tmp, 1 ) < 0) {
              perror("Writing Breakenv");
```

```
            error("be\n", 1);
        }
        more = 1;
        break;
    case BREAKTM:
        sprintf(tmp, "bt %s\n", (char *) str);
        if (MessageWrite( tmp, 1 ) < 0) {
            perror("Writing Break time");
            error("bt\n", 1);
        }
        more = 1;
        break;
    case RESET:
        if (MessageWrite( (char *) "quit\n", 0 ) < 0 ) {
            perror("Writing quit\reset command");
            error("Quit command\n", 1);
        }
        more = 1;
        break;
    case COMMAND:
        if (MessageWrite( (char *) str, 0 ) < 0 ) {
            perror("Writing command line");
            error("Line Command\n", 1);
        }
        if (strcmp( str, "quit\n" ) == 0 )
            resetmanager();
        more = 1;
        break;
    default:
        sprintf(tmp, "Unknown option in sendMsg %d (%d)", sendcomm, COMCOUNT);
        error(tmp, 3);
    }
}

/* so far doesnt really do anything useful */
void interface_init_screen(scr1, scr2, scr3)
    Widget scr1, scr2, scr3;
{
    mainscreen = scr1;
    subscreen = scr2;
    helpscreen = scr3;
}

/* used primarily by loadprocess to put the lines into a buffer than
   send it to screenbuffer all at once so no 'obvious' scrolling can be seen
   and slow down can be avoided. About the same as using "diskfile"
   type == 0 means just want to insert a line, otherwise lines are dump into
   screen buffer */
int putList2( str, type )
    char *str;
    int type;
{
    static int pos, cur, i, init = 0;
    extern int maxdisp;

    if (type == 0) {
        if (init == 0) {
            loadfile = CALLOC(maxdisp, char);
            init = 1;
            pos = 0;
            cur = lastpos;
        }
        i = strlen(str);
        for(; *str != '\0' && pos < maxdisp; pos++) loadfile[pos] = *(str++);
        return cur+pos-i+1;
    } else {
        XtTextPosition start;
        XtTextBlock tb;

        start = lastpos;
        tb.firstPos = 0;
        tb.length = pos;
        tb.ptr = loadfile;
        i = start + pos;
        XtTextReplace(subscreen, (XtTextPosition) start, (XtTextPosition) i, &tb);
        lastpos = XtTextGetInsertionPoint(subscreen);
        init = 0;
        free(loadfile);
        return 0;
    }
}

/* putList:
   puts str into the List window and returns the position it is in */
int putList( str, type )
    char *str;
    int type;
{
    XtTextPosition startpos;
    XtTextBlock tb;
    int i, oldstartpos;

    startpos = XtTextGetInsertionPoint(subscreen);
    tb.firstPos = 0;
    tb.length = strlen(str);
    tb.ptr=str;
    i = startpos + tb.length;
    oldstartpos = startpos;
    XtTextReplace(subscreen, (XtTextPosition) startpos,
                  (XtTextPosition) i, &tb);
    startpos = XtTextGetInsertionPoint(subscreen);
    lastpos = startpos;
    return oldstartpos+1;
}

/* reposition cursor on the listwindow */
void reposition( line )
    int line;
{
    XtTextSetInsertionPoint( subscreen, linestart[line]-1 );
}

/* putMain:
   puts str into the mainwindow and returns position */
int putMain( str )
    char *str;
{
    XtTextPosition startpos;
    XtTextBlock tb;
    int i;
    Cardinal args;
    Arg arg[3];

    startpos = XtTextGetInsertionPoint(mainscreen);
    tb.firstPos = 0;
    tb.length = strlen(str);
    tb.ptr=str;
    i = startpos + tb.length;
    XtTextReplace(mainscreen, (XtTextPosition) startpos,
                  (XtTextPosition) i, &tb);
    startpos += tb.length + 1;
    return startpos - tb.length - 1;
```

```
/* the do process loop patched into Xt to handle specific simulator returns */
void MainDo()
{
   int nb, val;
   char buf[1024];

   while(more) {                              /* if we await a simulator return */
         if ((nb = MessageRead( buf, 1024)) < 0) { /* read a line */
            perror("Reading from simulator");
            exit(1);
         }
         if (nb == 0) more = 0;  /* if nothing to read, or small error */
         else {
            if (instr(buf, "loaded ") != -1) {
                  int tmp;
                  for(tmp=nb-1; tmp >= 0 && isspace(buf[tmp]); tmp--);
                  /* must wait til we have the whole string */
                  while (buf[tmp] != PROMPT) {
                     if ( (tmp = MessageRead( buf+nb, 1024-nb )) < 0 ) {
                           perror("Reading from simulator");
                           exit(1);
                     }
                     nb += tmp;
                     for(tmp=nb-1; tmp >= 0 && isspace(buf[tmp]); tmp--);
                  }
#ifdef DEBUG
                  error(buf, 3);
#endif
                  loadprocess(buf);
            }
            if (nb > 0) { /* if there's something read */
                  for(nb--; nb >= 0 && isspace(buf[nb]); nb--);
                  putMain( buf );  /* put it in the main window */
                  /* if the simulator returns current instruction, move there */
                  if (instr(buf, " (P") >= 0 && (val = instr(buf, "0x")) >= 0) {

                     int i, addr;
                     char *tmp;
                     addr = hextol(buf+val+2);
                     for (i=0; i < linecount && addr >= line[i]; i++);
                     if (i < linecount) { /* posn the right line for display */
                           if (i+1 < linecount) reposition(i+1);
                           reposition(i);
                     }
                  }
                  if (buf[nb] == PROMPT) { /* if end of read, put a newline */
                     more--;
                     Finalpos = XtTextGetInsertionPoint(mainscreen);
                     if (more == 0) { /* at the end, upd the proc/task win */
                           for(nb=0; nb < NUMPROC; nb++)
                              if (processor[nb] != 0) updateProc( nb );
                           for(nb=0; nb < NUMTASK; nb++)
                              if (task[nb] != 0) updateTask( nb );
                     }
                  } else if (buf[nb] == '?') { /* if error message, reply */
                     MessageWrite("y\n", 0);
                     error("Top level question", 3);
                     strcat(buf, "y\n");
                  }
            }
         }
   }
}
```

interface.h, page 1

```
/* the interface definition headers, defines what is recognized as "end prompt", the
   number of Processor/tasks allowed, and enums the kind of commands passed to sendMsg,
   and the modules which are defined for both handler and interface modules */

#define PROMPT '>'

#define NUMPROC 8
#define NUMTASK 20


#define LOAD 0  /* "LOAD" MUST ALWAYS BE ZERO */
#define STEP (LOAD + 1)
#define RUN (STEP + 1)
/* #define CONT (RUN + 1) */
#define PROCESSOR (RUN + 1)
#define TASK (PROCESSOR + 1)
#define BREAKENV (TASK + 1)
#define BREAKTM (BREAKENV + 1)
#define RESET (BREAKTM + 1)
#define COMMAND (RESET + 1)
#define COMCOUNT (COMMAND + 1)


#define RUNSIZE 10

extern void sendMsg();
extern void interface_init_screen();
extern int putList(), putMain();
extern void MainDo();
extern void dogetproc(), dogettask();
extern int MessageRead();
extern char *dialog();
```

main.c, page 1

```
/*
 * Copyright (c) 1989 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that this notice is preserved and that due credit is given
 * to the University of California at Berkeley. The name of the University
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission. This software
 * is provided "as is" without express or implied warranty.
 */

#ifndef lint
static char copyright[] =
"@(#) Copyright (c) 1989 Regents of the University of California.\n\
 All rights reserved.\n";
#endif /* not lint */

#ifndef lint
static char sccsid[] = "@(#)main.c        4.1  9/11/89";
#endif /* not lint */

/* multix: multi-processor simulation X-interface utility.
 *
 *      A visual graphics interface written with X support for
 *      a graphical display of the simulator and it's processors and tasks.
 *
 *      Written by Pang (Swee Chee), (speaker@pisces)
 *
 */

#include <X11/Xlib.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <stdio.h>
#include <strings.h>
#include <signal.h>
#include <errno.h>

#include "defaults.h"
#include "general.h"

char title[MAXCHAR];

char *simulator;
int child;
int sv[2];

/* CHILD interrupt kill process */

killChild()
{
   MessageWrite("quit\n", 0);
   if ( kill(child, SIGKILL) != 0 )
         perror("Kill");
   close(sv[0]);
}

bombed(sig, code, scp)
    int sig, code;
    struct sigcontext *scp;
{
   extern void killWindows();
   killChild();
   killWindows();
   fprintf(stderr, "Error: %d (%d)\n", sig, code);
   exit(sig);
}

sendit(sig, code, scp)
    int sig, code;
    struct sigcontext *scp;
{
   extern int MessageWrite();
   extern int more;

   if (MessageWrite("\003\n", 1) < 0) {
         error("Writing for Ctrl-c", 3);
         return;
   }
   more = 1;
}

/*
```

```
*/
main(argc, argv)
    int argc;
    char **argv;
{
    char *args[MAXARG], *strind, *simulatorName, *loadfile, tmp[MAXCHAR];
    int i, nargs = 0, setsize, pass;
    void startsplit();

    args[nargs] = CALLOC( strlen(argv[0]) + 1, char );
    (void) strcpy( args[nargs++], *argv );


    simulatorName = SimulatorName;

    /* parse command fpath */

    setdisp('m', "");
    setdisp('p', "");
    setdisp('t', "");
    simulator = Simulator;
    loadfile = NULL;
    setsize = 1;

    for (argc--, argv++; argc > 0; argc--, argv++) {

        if (*argv[0] == '-') {
            switch(argv[0][1]) {
            case 'h':
                printHelp();
                error("", 0);
                break;
            case 'm':
                if (argv[0][2] == 'e') {
                    setsize = atoi(*(++argv));
                    argc--;
                    break;
                }
            case 'p':
            case 't':
                setdisp(argv[1], *(++argv));
                argc--;
                break;
            case 's':
                simulator = *(++argv);
                argc--;
                break;
            case 'e':
                pass = argc;
                argc = 0;
                break;
            }
        } else {
            if ( index(*argv, ':') != NULL ) setdisp('a', *argv);
            else {                          /* check if it is w_code file */
                if((*argv)[strlen(*argv)-1] == 'w' &&
                  (*argv)[strlen(*argv)-2] == '.')
                    loadfile = *argv;
                else {
                    if (nargs == MAXARG) error("Too many arguments", 2);

                    args[nargs] = CALLOC( strlen(*argv)+1, char );
                    (void) strcpy( args[nargs++], *argv );
                }
            }
        }
    }
    dispsize(setsize);
    startsplit();
    strcpy(tmp, simulator);
    for(pass--; pass>0; pass--) {
            strcat(tmp, " ");
            strcat(tmp, *(argv++));
    }
    strcat(tmp, "\n");
    error(tmp, 3);
    (void) MessageWrite( tmp, 0 );
    manager(simulatorName, loadfile, args, nargs);

}

void getport()
{
    char **sim_arg;

    child = fork();
    signal(SIGINT, sendit);             /* set up INT to call bombed */
    signal(SIGSEGV, bombed); /* and SEGV, and several others if you like */
    signal(SIGQUIT, bombed);
    signal(SIGTERM, bombed);

    switch( child ) {
            /* refer IPC Adv. PS1:8-26*28 */
    case -1:
            error("fork child", 1);
    case 0:                             /* CHILD SIDE */

            close(sv[0]);
            dup2(sv[1], 0);             /* stdin */
            dup2(sv[1], 1);             /* stdout */
            dup2(sv[1], 2);             /* stderr */
            if (sv[1] > 2)
              close(sv[1]);
            if (execlp(SHELL, "-e", SHELLPROG, 0) == -1)
              error("Can't execute simulator", 1);
            break;                      /* this is never reached, I hope */

    default:                            /* PARENT SIDE (parasite) */
            close(sv[1]);
            break;
    }
    resetmanager();
}

void startsplit()
{
    char c, *fpath = "/dev/ptyXX";
    int i;

    for(c='p', sv[0]= (-1); sv[0] <= 2 && c<='s'; c++) {
            struct stat statbuf;

            fpath[sizeof("/dev/pty")-1] = c;
            fpath[sizeof("/dev/ptyp")-1] = '0';

            if (stat(fpath, &statbuf)<0) break;
            for(i=0;i<16;i++) {
              fpath[sizeof("/dev/ptyp")-1]="0123456789abcdef"[i];
              sv[0]=open(fpath, O_RDWR, 0666);
              if (sv[0] > 2) break;
            }
```

```
    }
    if (sv[0] <= 2)
          error("Network ports can't open", 1);

    fpath[sizeof("/dev/")-1]='t';

    sv[1] = open(fpath, O_RDWR, 0666);
    if (sv[1] <= 2) error("Slave open error", 1);
    getport();
}

int MessageRead( buf, n )
    char *buf;
    int n;
{
    int tmp;
    char str[MAXCHAR];

    bzero(buf, n);
    bzero(str, MAXCHAR);
    tmp = read( sv[0], str, n );
    strncpy(buf, str, tmp);
    if (tmp == n) buf[--tmp] = '\0';
/*  printf("READ: %d %d %d\n %s \n ****\n %s \n *******", tmp, strlen(str), strlen(buf), buf, str); */
/*  printf("%d: %s\n", tmp, &buf[tmp-3]); */
    return tmp;
}

int MessageWrite( buf, type )
    int type;
    char *buf;
{
    int tmp, size;
    char str[MAXCHAR];

    size = strlen(buf);
    bzero(str, MAXCHAR);
    strcpy(str, buf);
    tmp = write( sv[0], str, size*sizeof(char));
/*  printf("RIGHT: %d\n %s \n ****\n %s \n *******", tmp, buf, str); */

    if (type == 1) putMain( buf );
    return tmp;
}
printHelp()
{ }
```

mainmenu.h, page 1

```
/* the Main menu: there are 3 parts to this menu */
#define MAINMENU 3

static char *mainmenu[MAINMENU][2] = {
  { "Processors", "The processor menu" },
  { "Tasks", "The Task Table Entries menu" },
  { "Summary", "The Summary information window" }
};

void procMain(), taskMain(), summMain();

static void (*mainmenufn[MAINMENU])() = {
   procMain, taskMain, summMain };
```

```
/*
 * Copyright (c) 1989 Regents of the University of California
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that this notice is preserved and that due credit is given
 * to the University of California at Berkeley. The name of the University
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission. This software
 * is provided "as is" without express or implied warranty.
 */

#ifndef lint
static char sccsid[] = "@(#)manager.c     4.1  9/11/89";
#endif /* not lint */

/* manager.c: support for multix.
 *      the main menu manager module is here.
 *      is in charge of the creation and placement of menu windows etc.
 */

#include <stdio.h>
#include <strings.h>
#include <ctype.h>
#include <signal.h>
#include <errno.h>
#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Box.h>
#include <X11/Command.h>
#include <X11/Dialog.h>
#include <X11/Label.h>
#include <X11/Load.h>
#include <X11/Scroll.h>
#include <X11/AsciiText.h>
#include <X11/VPaned.h>
#include <X11/Viewport.h>
#include <X11/Cardinals.h>

#include "MenuBox/MenuBox.h"
#include "MenuBox/MenuShell.h"
#include "MenuBox/Menu.h"
#include "defaults.h"
#include "general.h"
#include "interface.h"
#include "menucmd.h"
#include "manager.h"
#include "mainmenu.h"

/* Command line options table.  Only resources are entered here...there is a
   pass over the remaining options after XtParseCommand is let loose. */

static char *listhead = "Listing Window <0.1>\n";
static char *debughead = "Debugging Window <0.2>\n";
static char *mtext, *list;
static char buf[MAXCHAR];
int curr, mainline, maxdisp, initialized = 0;
extern int Finalpos;
Widget whelp, mainscreen, subscreen;

char *menudpy, *procdpy, *taskdpy;

void dispsize(size)
    int size;
{
  maxdisp = max(1, size) * MAXHISTORY;
  {
        char temp[80];
        sprintf(temp, "Maxdisp: %d (%d)\n", maxdisp, size);
        error(temp, 3);
  }
  init_interface(size, 0);
  mtext = CALLOC(maxdisp, char);
  list = CALLOC(maxdisp, char);
}

void resetmanager()
{
  curr = -10;
  if (initialized) {
        extern int lastpos;
        TextWidget sb = (TextWidget) subscreen;
        XtTextBlock blk;

        blk.firstPos = 0;
        blk.length = 0;
        blk.ptr = "";

        fprintf(stderr, "ResetManager: %d\n", lastpos);

        XtTextReplace(subscreen, 0, lastpos, &blk);
        (void) putList(listhead, 0);
  }
  (void) handler_init((Widget) NULL);
  (void) init_interface(0, 1);
}

/* getlistposn: returns the current position of the "insertion cursor" */
int getlistposn()
{
  return XtTextGetInsertionPoint(subscreen);
}

/* setdisp: currently doesnt do anything fantastic */
setdisp(cmd, dpy)
   char cmd, *dpy;
{
  switch(cmd) {
  case 'm':
        menudpy = dpy;
        break;
  case 'p':
        procdpy = dpy;
        break;
  case 't':
        taskdpy = dpy;
        break;
  case 'a':
        if (menudpy[0] = '\0') menudpy = dpy;
        if (procdpy[0] = '\0') procdpy = dpy;
        if (taskdpy[0] = '\0') taskdpy = dpy;
        break;
  default:
        error("Unknown option to setdisp", 2);
  }
}

/*
```

```
 */
/* Editing commands */

XtActionProc SelWord0(w, event, parm, num)
    Widget w;
    XEvent *event;
    String *parm;
    Cardinal *num;
{
   char s[MAXCHAR], *text;
   int pos, left, right;

   if (w == mainscreen) text = mtext;
   else text = list;
   pos = XtTextGetInsertionPoint(w);
   for(left=pos; left > 0 && (isspace(text[left]) || text[left] == '\n'); left--);
   for(right=(++left); right < maxdisp && text[right] != '\0' &&
                    !isspace(text[right]); right++);
   if (left == (--right) ) return;
   strncpy(s, text+left, right-left+1);
   XtTextUnsetSelection(w);
   XStoreBytes(XtDisplay(w), s, right-left+1);
   XtTextSetSelection(w, left, right+1);


XtActionProc ClrSel(w, event, parm, num)
    Widget w;
    XEvent *event;
    String *parm;
    Cardinal *num;
{
   XStoreBytes(XtDisplay(w), NULL, 0);
   XtTextUnsetSelection(w);
}

XtActionProc DelChar(w, event, parm, num)
    Widget w;
    XEvent *event;
    String *parm;
    Cardinal *num;
{
   TextWidget wt = (TextWidget) w;
   XtTextBlock blk;
   int pos;

   pos = XtTextGetInsertionPoint(w);
   XtTextSetInsertionPoint(w, pos);
   if ( pos > Finalpos ) {
      blk.firstPos = 0;
          blk.length = 0;
          blk.ptr = "";
          XtTextReplace(w, pos-1, pos, &blk);
   }
}

XtActionProc DelWord(w, event, parm, num)
    Widget w;
    XEvent *event;
    String *parm;
    Cardinal *num;
{
    XtTextBlock blk;
    int pos, left;

    blk.firstPos = 0;
    blk.length = 0;
    blk.ptr = "";

    pos = XtTextGetInsertionPoint(w);
    for(left=pos; left > Finalpos && !isspace(mtext[left]); left--);
    if (left==pos) return;
    XtTextReplace(w, left, pos, &blk);
}

XtActionProc DelLine(w, event, parm, num)
    Widget w;
    XEvent *event;
    String *parm;
    Cardinal *num;
{
    XtTextBlock blk;
    int i, pos;

    blk.firstPos = 0;
    blk.length = 0;
    blk.ptr = "";

    pos = XtTextGetInsertionPoint(w);
    for(i=pos; i > Finalpos && mtext[i] != '\n'; i--);
    XtTextReplace(w, i, pos, &blk);
}

XtActionProc SendCmd(w, event, parm, num)
    Widget w;
    XEvent *event;
    String *parm;
    Cardinal *num;
{
    int i;
    char *s, p[MAXCHAR];

    s = mtext+Finalpos;
    if ((i = strlen(s)) > 0 && s[i-1] != '\n')
            fprintf(stderr, "NO newline!\n");
    strcpy(p, s);
    sendMsg(COMMAND, p, 0);
}

XtActionProc SigInt(w, event, parm, num)
    Widget w;
    XEvent *event;
    String *parm;
    Cardinal *num;
{
    sendMsg(COMMAND, "\003\n", 0);
}

/* MAKEMENU: makes the Main Menu Widgets */
static void makemenu( top , name)
    Widget top;
    char *name;
{
    static XtActionsRec tbl[] = {
            {"SigInt", (XtActionProc) SigInt},
            {"SelWord0", (XtActionProc) SelWord0},
            {"ClrSel", (XtActionProc) ClrSel},
            {"DelChar", (XtActionProc) DelChar},
            {"DelWord", (XtActionProc) DelWord},
            {"DelLine", (XtActionProc) DelLine},
            {"SendCmd", (XtActionProc) SendCmd},
            {NULL, NULL},
    };
```

```
static String trans = "\
        Ctrl<Key>U:       DelLine()\n\
        Ctrl<Key>C:       SigInt()\n\
        Ctrl<Key>W:       DelWord()\n\
        Ctrl<Key>H:       DelChar()\n\
        <Key>Delete:      DelChar()\n\
        <Key>BackSpace:   DelChar()\n\
        <Key>Return:      end-of-file() newline() SendCmd() end-of-file()\n\
        <Key>:            end-of-file() insert-char()\n\
        <FocusIn>:        focus-in()\n\
        <FocusOut>:       focus-out()\n\
        <Btn1Down>:       select-start() ClrSel()\n\
        <Btn1Motion>:     extend-adjust()\n\
        <Btn1Up>:         extend-end(PRIMARY, SelWord0)\n\
        <Btn1Up>(2):      SelWord0()\n\
        <Btn2Down>:       end-of-file() insert-selection(PRIMARY, SelWord0)";

Arg arg[MAXARG];
Cardinal args;
Widget title, label, box, temp;
XFontStruct *ft;
Int i;
char s[80];


/* TITLE */

title = XtCreateManagedWidget("vpane", vPanedWidgetClass, top,
                              vpane_args, XtNumber(vpane_args));

If ((ft = XLoadQueryFont(XtDisplay(top), DEFAULT_TITLE_FONT)) == NULL &&
        (ft = XLoadQueryFont(XtDisplay(top), "gallant.r.19")) == NULL &&
        (ft = XLoadQueryFont(XtDisplay(top), "9x15")) == NULL &&
        (ft = XLoadQueryFont(XtDisplay(top), "fixed")) == NULL) {
        error("No font for title", 0);
}


larger_font[0].value = (XtArgVal) ft;
label = XtCreateManagedWidget(name, labelWidgetClass,
                              title, larger_font, XtNumber(larger_font));
XtPanedAllowResize(label, True);

{
        Int i=0;
        PopupMenu *menu;

        menu = MenuCreate( top, label, "CONTROL" );
        MenuBind(label, "CONTROL", "<EnterNotify>");

        for( i=0; i < MAINMENU; i++)
          MenuAddSelection(menu, mainmenu[i][0], mainmenu[i][1], mainmenufn[i], help, NULL);

        MenuReady( menu );
        handler_init(top);
}
/* HELP/COMMAND WINDOW */

whelp = XtCreateManagedWidget("Help Window", labelWidgetClass,
                              title, help_args, XtNumber(help_args));
XtPanedAllowResize(whelp, False);

XtAddEventHandler(whelp, EnterWindowMask, 0, help, (caddr_t) "Help Window");
XtAddEventHandler(whelp, LeaveWindowMask, 0, help, (caddr_t) "  ");
```

```
/* LISTING WINDOW */

list[0] = '\0';
args = 0;

XtSetArg(arg[args], XtNheight, 300); args++;
XtSetArg(arg[args], XtNwidth, 500); args++;
XtSetArg(arg[args], XtNlength, MAXHISTORY); args++;
XtSetArg(arg[args], XtNstring, list); args++;
XtSetArg(arg[args], XtNallowResize, TRUE); args++;
XtSetArg(arg[args], XtNtextOptions, scrollVertical); args++;
XtSetArg(arg[args], XtNleftMargin, 2); args++;
XtSetArg(arg[args], XtNtranslations, XtParseTranslationTable(trans)); args++;
XtSetArg(arg[args], XtNeditType, XttextEdit); args++;

subscreen = XtCreateManagedWidget("List Window", asciiStringWidgetClass,
                                  title, arg, args);

XtAddEventHandler(subscreen, EnterWindowMask, 0, help,
                  (caddr_t) "Listing Window <0.1>");
XtAddEventHandler(subscreen, LeaveWindowMask, 0, help, (caddr_t) "  ");
XtAddActions(tbl, XtNumber(tbl));

/* BUTTONS */

box = XtCreateManagedWidget("commands", boxWidgetClass, title, 0, 0);
XtPanedAllowResize(box, False);

for( i=0; i < MAXCOMMAND; i++) {
        static XtCallbackRec callback[2];

        args = 0;
        callback[0].callback = command_function[i];
        XtSetArg(arg[args], XtNcallback, callback); args++;

        temp = XtCreateManagedWidget(command[i][0], commandWidgetClass, box, arg, args);
        XtAddEventHandler(temp, EnterWindowMask, 0, help, (caddr_t) command[i][1]);
        XtAddEventHandler(temp, LeaveWindowMask, 0, help, (caddr_t) "  ");
}

/* Main Display Window */

mtext[0] = '\0';
args = 0;

XtSetArg(arg[args], XtNheight, 400); args++;
XtSetArg(arg[args], XtNwidth, 500); args++;
XtSetArg(arg[args], XtNlength, MAXHISTORY); args++;
XtSetArg(arg[args], XtNstring, mtext); args++;
XtSetArg(arg[args], XtNtextOptions, scrollVertical); args++;
XtSetArg(arg[args], XtNleftMargin, 2); args++;
XtSetArg(arg[args], XtNtranslations, XtParseTranslationTable(trans)); args++;
XtSetArg(arg[args], XtNeditType, XttextEdit); args++;

mainscreen = XtCreateManagedWidget("Text Window", asciiStringWidgetClass,
                                   title, arg, args);

XtAddEventHandler(mainscreen, EnterWindowMask, 0, help,
                  (caddr_t) "Debugging Window <0.2>");
XtAddEventHandler(mainscreen, LeaveWindowMask, 0, help, (caddr_t) "  ");
XtAddActions(tbl, XtNumber(tbl));

interface_init_screen(mainscreen, subscreen, whelp);
(void) putList(listhead, 0);
(void) putMain(debughead);
mainline = XtTextGetInsertionPoint(mainscreen);
```

```
    initialized = 1;
}

/* MANAGER: main menu startup and main graphics loop */

manager( title, file, argv, argc )
    char *title, *file, **argv;
    int argc;
{
    Widget topmenu, topproc, toptask;
    XEvent event;

    topmenu = XtInitialize( title, "MENU", options, XtNumber(options),
                                    &argc, argv);
    makemenu( topmenu, title );

    XtRealizeWidget(topmenu);
    if (file != NULL)
            sendMsg( LOAD, file, 1); /* "0" is code for "LOAD" and must always be so */

    /* XtMainLoop plus the simulator server */

    while(XtPending()) {
            XtNextEvent(&event);
            XtDispatchEvent(&event);
    }

    forever {
            XtNextEvent(&event);
            XtDispatchEvent(&event);
            MainDo();
    }
}
```

manager.h, page 1

```
/* just some basic Xt parameters */
static    Arg       vpane_args[] = {
   {XtNallowResize, (XtArgVal) True},
};

static    Arg       larger_font[] = {
   {XtNfont,                  0},
};

static    Arg       help_args[] = {                                                    "},
   {XtNlabel, (XtArgVal) "
};

static XrmOptionDescRec options[] = {
   {"-scroll", "scroll",      XrmoptionNoArg, "True"},
   {"-stretch","stretch",     XrmoptionNoArg, "True"},
   {"-label",   XtNlabel,     XrmoptionSepArg,          NULL}
};

extern void help();
```

menucmd.h, page 1

```
/* defines the kind of commands to be displayed on the "command" window, with
   the issuing "help" messages, and the function the button triggers */

#define MAXCOMMAND 9 /* number of commands on the window is currently 8 */

extern void editor(); /* handles the main window's keyboard action for transmit */
extern void handler_init(); /* performs initialization necessary for the module */
/* below lists the function to be called */
extern void quit(), step(), load(), run(), breakenv(), breakpoint(), breaktime(), config(), reset();

/* this array contains the "command", "help mssg" of all the buttons in the
   command window */
static char *command[MAXCOMMAND][2] = {
    {"Load","Loads the byte compiled program"},
    {"Step","Steps through the program"},
    {"Run","Runs the program (Note: no update is performed)"},
    {"Breakenv","Set environment for break"},
    {"Breakpoint","Set/Delete breakpoints"},
    {"Breaktime","Set/Delete breaktimes"},
    {"Config","Change certain settings"},
    {"Reset","Resets Nusim, so you can start afresh"},
    {"Quit","Quits from MULTIX.. clear enough"},
};

/* defines the function they call, in the same order as the above array */
static void (*command_function[MAXCOMMAND])() = {
    load, step, run, breakenv, breakpoint, breaktime, config, reset, quit,
};
```

```
/*
 * Copyright (c) 1989 Regents of the University of California
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that this notice is preserved and that due credit is given
 * to the University of California at Berkeley. The name of the University
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission. This software
 * is provided "as is" without express or implied warranty.
 */

#ifndef lint
static char sccsid[] = "@(#)misc.c          4.1   9/11/89";
#endif /* not lint */

/* misc.c: support for multix.
 *      miscellaneous functions for processor/tasks control are here.
 *      is in charge of the creation and update of these windows etc.
 */

#include <stdio.h>
#include <strings.h>
#include <ctype.h>
#include <errno.h>
#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Box.h>
#include <X11/Command.h>
#include <X11/Dialog.h>
#include <X11/Label.h>
#include <X11/Load.h>
#include <X11/Scroll.h>
#include <X11/AsciiText.h>
#include <X11/VPaned.h>
#include <X11/Viewport.h>
#include <X11/Cardinals.h>
#include <X11/Shell.h>

#include "MenuBox/MenuBox.h"
#include "MenuBox/MenuShell.h"
#include "MenuBox/Menu.h"
#include "defaults.h"
#include "general.h"
#include "interface.h"
#include "processor.h"

typedef char *Mystring;
extern char *needline();

Widget top;
static    Arg       larger_font[] = {
    {XtNfont,                   0},
};

Widget *taskconfig, taskmain, *procconfig, procmain; /* holds the main menu windows */
Widget procset[NUMPROC], *procreg[NUMPROC], /* "set" holds active proc/task windows */
      taskset[NUMPROC], *taskreg[NUMPROC]; /* "reg" holds each of the reg */
Mystring *flatproc[NUMPROC], *flattask[NUMPROC]; /* contains the actual name of disp reg */

/* format:
   places "Name : Value" into Label padded to Max number of spaces. */
static void format( label, name, val )
    char *label, *name, *val;
{
  int j;
  char tmp[MAXWORDSIZE+3];

  if (val == NULL) return;
  for(j=0; j<MAXWORDSIZE+2; j++) tmp[j] = ' ';
  tmp[j] = '\0';
  for(j=1+MAXWORDSIZE - strlen(val); j<MAXWORDSIZE+1; j++) tmp[j] = *(val++);

  label[0] = ' ';
  strcpy(&label[1], name);
  for(j=strlen(label); j <= MAXLEN+1; j++) label[j] = ' ';
  label[j++] = ':';
  label[j++] = '\0';
  strcat(label, tmp);
}

/* MANAGE: start or destroy proc/task window */

void manageProc(n, top)
    int n;
    Widget top;
{
  if (procset[n] == (Widget) NULL) { /* if it is not being displayed, display it */
        char s[80];
        int i, j, k, tmp;
        XFontStruct *ft;
        Widget pane, title, box;

        tmp = MAXPROC + PROCVAR*(MAXNUM - 1); /* max size needed for all reg */
        procreg[n] = CALLOC( tmp, Widget ); /* widget for each reg */
        flatproc[n] = CALLOC( tmp, Mystring); /* name of each reg */

        for( i=0, j=0; i < MAXPROC; i++ )
          switch(procstat[i]) {
          case -1: /* inactive, ignore */
                break;
          case 0: /* active, display it */
                flatproc[n][j] = (Mystring) proc[i];
                j++;
                break;
          default: /* variable type: display only active of variable set */
                for(k=0; k < MAXNUM; k++)
                  if (procvar[procstat[i]-1][k] == 0) {
                        sprintf(s, "%s%d", proc[i], k);
                        flatproc[n][j] = (Mystring) CALLOC( strlen(s)+1, char );
                        strcpy((char *) flatproc[n][j], s);
                        j++;
                  }
          }
        if (j < MAXPROC) flatproc[n][j] = (Mystring) NULL; /* end of widget set */

        sprintf(s, "processor%d", n);
        procset[n] = XtCreatePopupShell(s, shellWidgetClass, top, NULL, 0);
        pane = XtCreateManagedWidget("processor", vPanedWidgetClass, procset[n],
                                     NULL, 0);
        sprintf(s, "    ==== PROCESSOR %2d ====    ", n);
        if ((ft = XLoadQueryFont(XtDisplay(top), DEFAULT_SUBTTL_FONT)) != NULL) {
          larger_font[0].value = (XtArgVal) ft;
          title = XtCreateManagedWidget(s, labelWidgetClass, pane, larger_font,
                                        XtNumber(larger_font));
        } else
          title = XtCreateManagedWidget(s, labelWidgetClass, pane, NULL, 0);
        box = XtCreateManagedWidget("procBox", boxWidgetClass, pane, NULL, 0);

        for( i=0; i < ; i++ ) {
```

```
            format(s, (char *) flatproc[n][i], "0");
            procreg[n][i] = XtCreateManagedWidget(s, labelWidgetClass, box, NULL, 0);
          }
          XtPopup(procset[n]);
          sendMsg(PROCESSOR, n, 1);
  } else { /* if progset[n] != NULL, was active, so destroy it */
          free((char *) flatproc[n]);
          free((char *) procreg[n]);
          XtDestroyWidget(procset[n]);
          procset[n] = (Widget) NULL;
          sendMsg(PROCESSOR, n, 0);
  }
}

void manageTask(n, top)
   int n;
   Widget top;
{
  /* for comments, compare above */
  if (taskset[n] == (Widget) NULL) {
          char s[80];
          int i, j, k, tmp;
          XFontStruct *ft;
          Widget pane, title, box;

          tmp = MAXTTE + TTEVAR*(MAXNUM - 1);
          taskreg[n] = CALLOC( tmp, Widget );
          flattask[n] = CALLOC( tmp, Mystring );

          for( i=0, j=0; i < MAXTTE; i++ )
            switch(ttestat[i]) {
            case -1:
                    break;
            case 0:
                    flattask[n][j] = (Mystring) tte[i];
                    j++;
                    break;
            default:
                    for(k=0; k < MAXNUM; k++)
                      if (ttevar[ttestat[i]-1][k] == 0) {
                              sprintf(s, "%s%d", tte[i], k);
                              flattask[n][j] = (Mystring) CALLOC( strlen(s)+1, char );
                              strcpy((char *) flattask[n][j], s);
                              j++;
                      }
            }
          if (j < MAXTTE) flattask[n][j] = (Mystring) NULL;

          sprintf(s, "task%d", n);
          taskset[n] = XtCreatePopupShell(s, shellWidgetClass, top, NULL, 0);
          pane = XtCreateManagedWidget("task", vPanedWidgetClass, taskset[n],
                                       NULL, 0);
          sprintf(s, "      ==== TASK %2d ====      ", n);
          if ((ft = XLoadQueryFont(XtDisplay(top), DEFAULT_SUBTTL_FONT)) != NULL) {
            larger_font[0].value = (XtArgVal) ft;
            title = XtCreateManagedWidget(s, labelWidgetClass, pane, larger_font,
                                          XtNumber(larger_font));
          } else
            title = XtCreateManagedWidget(s, labelWidgetClass, pane, NULL, 0);
          box = XtCreateManagedWidget("taskBox", boxWidgetClass, pane, NULL, 0);

          for( i=0; i < j; i++ ) {
            format(s, (char *) flattask[n][i], "0");
            taskreg[n][i] = XtCreateManagedWidget(s, labelWidgetClass, box, NULL, 0);
          }
          XtPopup(taskset[n]);
          sendMsg(TASK, n, 1);
  } else {
          free((char *) flattask[n]);
          free((char *) taskreg[n]);
          XtDestroyWidget(taskset[n]);
          taskset[n] = (Widget) NULL;
          sendMsg(TASK, n, 0);
  }
}

/* UPDATE: updates the values in the registers of proc/tte */

void updateProc( n )
   int n;
{
  static char label[MAXCHAR];
  static Arg arg[] = { {XtNlabel, (XtArgVal) label } };
  int i, j, k, dummy = 0;
  char *p, s[MAXCHAR], reg[MAXLEN];

  if (procset[n] == (Widget) NULL) return; /* call on inactive proc widget */

  sprintf(s, "ps %d\n", n); /* command to send is "ps <processor number>" */
  if ( MessageWrite(s, 0) < 0 ) {
          perror("Writing ps");
          error(s, 1);
  }

  /* READ from simulator: assume is of type
     {{{' "REG':" "VAL' "}*rubbish*}*\n}* */
  while( (p = readline( dummy++ )) != NULL )
          while ((i = inchr(p, ' : ')) != -1) {
            p[i] = '\0';
            for(; *p != '\0' && isspace(*p); p++);
            if (*p != NULL) { /* search whether the REG found is being displayed */
                    for(j=0; flatproc[n][j] != (Mystring) NULL &&
                      strcmp(p, (char *) flatproc[n][j]) != 0; j++);
                    if (flatproc[n][j] != (Mystring) NULL) { /* if it is displayed */
                      p += i+1;
                      for(;*p != '\0' && isspace(*p); p++);
                      for(k=0;p[k] != '\0' && !isspace(p[k]); k++);
                      p[k] = '\0';
                      format(label, (char *) flatproc[n][j], p); /* update value */
                      p += k+1;
                      XtSetValues(procreg[n][j], arg, XtNumber(arg));
                    }
            }
          }
}


void updateTask( n )
   int n;
{
  /* for comments, see above */
  static char label[MAXCHAR];
  static Arg arg[] = { {XtNlabel, (XtArgVal) label } };
  int i, j, k, dummy = 0;
  char *p, s[MAXCHAR], reg[MAXLEN];

  if (taskset[n] == (Widget) NULL) return;

  sprintf(s, "tte %d\n", n);
  if ( MessageWrite(s, 0) < 0 ) {
          perror("Writing ps");
          error(s, 1);
```

```
    }

    while( (p = needline( dummy++ )) != NULL )
            while ((i = inchr(p, ' : ')) != -1) {
                p[i] = '\0';
                for(; *p != '\0' && isspace(*p); p++);
                if (*p != NULL) {
                        for(j=0; flattask[n][j] != (Mystring) NULL &&
                            strcmp(p, (char *) flattask[n][j]) != 0; j++);
                        if (flattask[n][j] != (Mystring) NULL) {
                            p += i+1;
                            for(;*p != '\0' && isspace(*p); p++);
                            for(k=0;p[k] != '\0' && !isspace(p[k]); k++);
                            p[k] = '\0';
                            format(label, (char *) flattask[n][j], p);
                            p += k+1;
                            XtSetValues(taskreg[n][j], arg, XtNumber(arg));
                        }
                }
            }
}

/* CONFIGURATION MODULE */

/* called when the "Quit config" button for that window is depressed */
void Killconfig(w, client, call)
    Widget w;
    caddr_t client, call;
{
    Widget *conf = (Widget *) client;
    XtDestroyWidget(*conf);
    *conf = (Widget) NULL;
}

/* called when the button for that "variable list" widget is pressed */
void SetVarReg(w, client, call)
    Widget w;
    caddr_t client, call;
{
    static Arg arg[2];
    Cardinal argn;
    int *varreg = (int *) client;

    *varreg = (-1) - *varreg; /* toggle */
    argn = 0;
    XtSetArg(arg[argn], XtNlabel, (*varreg == 0)?"      ON      ":"      OFF      "); argn++;
    XtSetValues( w, arg, argn );
}

/* called when the button for the variable register is pressed */
void ModifyVarReg(w, client, call)
    Widget w;
    caddr_t client, call;
{
    extern void help();
    PopupMenu *menu = NULL;
    int i, *varreg = (int *) client;

    menu = MenuCreate( top , w, "VariableList" );
    for(i=0; i< MAXNUM; i++)
            MenuAddSelection( menu,(varreg[i] == 0)?"      ON      ":"      OFF      ", "",
                              SetVarReg, help, &varreg[i]);
    MenuReady( menu );
    XtPopup(menu->shell);
}

/* called when the button for the processor regiser is pressed */
void ModifyProcReg(w, client, call)
    Widget w;
    caddr_t client, call;
{
    static char label[MAXCHAR];
    static Arg arg[] = { {XtNlabel, (XtArgVal) label } };
    int i = (int) client;

    if (procstat[i] > 0) error("Read wrong i in modifyprocreg", 3);
    procstat[i] = (-1) - procstat[i];
    sprintf(label, "%-6s is %-3s", proc[i], (procstat[i]==0)?"ON":"OFF");
    XtSetValues( procconfig[i+1], arg, XtNumber(arg) );
}

/* called by "handler" to manage configuration window for processor */
void configProc(sendtop)
    Widget sendtop;
{
    int i, j, k, size = MAXPROC + 1;
    Widget title, pane, box;
    Arg arg[1];
    Cardinal argn;
    char tmp[MAXCHAR];

    top = sendtop;
    if (procmain == (Widget) NULL) { /* if config window for proc not activated */
            procconfig = CALLOC( size, Widget ); /* activate it */
            procmain = XtCreatePopupShell("ProcConfig", shellWidgetClass, top, NULL, 0);
            pane = XtCreateManagedWidget("procc", vPanedWidgetClass, procmain, NULL, 0);
            title = XtCreateManagedWidget(" CONFIGURE : PROC ", labelWidgetClass,
                                          pane, NULL, 0);
            box = XtCreateManagedWidget("procBox", boxWidgetClass, pane, NULL, 0);

            argn = 0;
            XtSetArg(arg[argn], XtNlabel, "Quit config"); argn++; /* first button kills */
            procconfig[0] = XtCreateManagedWidget( "command", commandWidgetClass,
                                                   box, arg, argn );
            XtAddCallback( procconfig[0], XtNcallback, Killconfig, &procmain );
            for(i=1; i < size; i++)
              switch(procstat[i-1]) {
              case -1:
              case 0: /* normal registers */
                    sprintf(tmp, "%-6s is %-3s", proc[i-1], (procstat[i-1]==0)?"ON":"OFF");
                    argn = 0;
                    XtSetArg(arg[argn], XtNlabel, tmp); argn++;
                    procconfig[i] = XtCreateManagedWidget( "command", commandWidgetClass,
                                                           box, arg, argn );
                    XtAddCallback( procconfig[i], XtNcallback, ModifyProcReg, (caddr_t) i-1 );
                    break;
              default: /* variable ones: slight differences */
                    sprintf(tmp, "%-6s var >>", proc[i-1]);
                    argn = 0;
                    XtSetArg(arg[argn], XtNlabel, tmp); argn++;
                    procconfig[i] = XtCreateManagedWidget( "command", commandWidgetClass,
                                                           box, arg, argn );
                    XtAddCallback( procconfig[i], XtNcallback, ModifyVarReg,
                                   (caddr_t) procvar[procstat[i-1]-1] );
              }
            XtPopup(procmain);
    } else { /* if already displayed, destroy it */
            free(procconfig);
            XtDestroyWidget(procmain);
            procmain = (Widget) NULL;
    }
```

```
/* these two are like the top two, but for task, seperated so it maybe possible to
   change one without affecting the other, not to assume they are equivalent for
   all simulators */

void ModifyTaskReg(w, client, call)
    Widget w;
    caddr_t client, call;
{
    static char label[MAXCHAR];
    static Arg arg[] = { {XtNlabel, (XtArgVal) label } };
    int i = (int) client;

    if (ttestat[i] > 0) error("Read wrong i in modifytaskreg", 3);
    ttestat[i] = (-1) - ttestat[i];
    sprintf(label, "%-6s is %-3s", tte[i], (ttestat[i]==0)?"ON":"OFF");
    XtSetValues( taskconfig[i+1], arg, XtNumber(arg) );


void configTask(sendtop)
    Widget sendtop;
{
    int i, j, k, size = MAXTTE + 1;
    Widget title, pane, box;
    Arg arg[1];
    Cardinal argn;
    char tmp[MAXCHAR];

    top = sendtop;
    if (taskmain == (Widget) NULL) {
            taskconfig = CALLOC( size, Widget );
            taskmain = XtCreatePopupShell("TaskConfig", shellWidgetClass, top, NULL, 0);
            pane = XtCreateManagedWidget("taskc", vPanedWidgetClass, taskmain, NULL, 0);
            title = XtCreateManagedWidget(" CONFIGURE : TASK ", labelWidgetClass,
                                          pane, NULL, 0);
            box = XtCreateManagedWidget("taskcBox", boxWidgetClass, pane, NULL, 0);

            argn = 0;
            XtSetArg(arg[argn], XtNlabel, "Quit config"); argn++;
            taskconfig[0] = XtCreateManagedWidget( "command", commandWidgetClass,
                                                   box, arg, argn );
            XtAddCallback( taskconfig[0], XtNcallback, Killconfig, &taskmain );
            for(i=1; i < size; i++)
               switch(ttestat[i-1]) {
               case -1:
               case 0:
                       sprintf(tmp, "%-6s is %-3s", tte[i-1], (ttestat[i-1]==0)?"ON":"OFF");
                       argn = 0;
                       XtSetArg(arg[argn], XtNlabel, tmp); argn++;
                       taskconfig[i] = XtCreateManagedWidget( "command", commandWidgetClass,
                                                              box, arg, argn );
                       XtAddCallback( taskconfig[i], XtNcallback, ModifyTaskReg, (caddr_t) i-1 );
                       break;
               default:
                       sprintf(tmp, "%-6s var >>", tte[i-1]);
                       argn = 0;
                       XtSetArg(arg[argn], XtNlabel, tmp); argn++;
                       taskconfig[i] = XtCreateManagedWidget( "command", commandWidgetClass,
                                                              box, arg, argn );
                       XtAddCallback(taskconfig[i], XtNcallback, ModifyVarReg,
                                     (caddr_t) ttevar[ttestat[i-1]-1] );
               }
            XtPopup(taskmain);
    } else {
            free(taskconfig);
            XtDestroyWidget(taskmain);
            taskmain = (Widget) NULL;
    }
}
```

```
/* DEFAULTS registors that will be listed on the Proc/Task lists are defined here */

/* User can change this and recompile */

#define MAXLEN 5 /* maximum length of the number of char in Registor name */
#define MAXNUM 8 /* maximum number of the variable types to display */

/* for PROCESSOR table */

#define MAXPROC 17 /* number of registors to display, even number suggested */
#define PROCVAR 2  /* number of registors with variable counts */

char *proc[MAXPROC] = { /* name of each registor to display, in order of appearance */
   "P", "CP", "E", "B", "TS", "TR", "H", "HB", "S", "mode", "cut", /* 11 */
   "PDL", "oP", "cflow", "timer", /* 4 */
   "t", "A", /* 2 */
};

int procstat[MAXPROC] = { /* status of particular reg: 0 - display, -1 - not display,
                             n - for n > 0 are the variable types */
    0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0,
   -1, 0, -1, -1,
    1, 2, /* corresponds to the list below "procvar" */
};

int procvar[PROCVAR][MAXNUM] = { /* -1 not displayed, 0 displayed */
    { -1, -1, -1, -1, -1, -1, -1, -1},
    { 0, 0, 0, 0, -1, -1, -1, -1},
};


/* for TASK table */

#define MAXTTE 22 /* number of registors to display, even number suggested */
#define TTEVAR 1

char *tte[MAXTTE] = { /* name of each registor to display, in order of appearance */
    "P", "CP", "E", "B", "TR", "H", "HB", "S", "cut", "mode", "state", /* 11 */
    "Wb", "HPb", "STKb", "TRLb", /* 4 */
    "par", "parb", "pari", /* 3 */
    "C0", "C1", "KC", /* 3 */
    "A", /* 1 */
};

int ttestat[MAXTTE] = {
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0,
    0, 0, 0,
    0, 0, 0,
     1,
};

int ttevar[TTEVAR][MAXNUM] = {
    { 0, 0, 0, 0, 0, -1, -1, -1, -1 },
};
```

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UCB/CSD 89/532	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION The Regents of the University of California 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION ONR
6c. ADDRESS (City, State, and ZIP Code) Berkeley, California 94720	7b. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Street Arlington, VA 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION DARPA 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-88-K-0579
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22209	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO.: DARPA; PROJECT NO.; TASK NO.; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

* XNUSIM - Graphical Interface for a Multiprocessor Simulator

12. PERSONAL AUTHOR(S)

* Swee-Chee Pang

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month,	15. PAGE COUNT
technical	FROM 07/01/88 TO 11/30/90	* September 1989	* 67

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Xnusim is an X11 Window Interface for the Multi-Processor simulator Nusim. It is a display oriented interface between the simulator and the user via UNIX[1] sockets with graphical objects such as menus, buttons, etc. It is designed in such a way that would allow it to be used with other simulators of the same class. This paper intends to describe the functionality of the objects, structures and program modules of XNUSIM in detail.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Andre M. Van Tilborg	22b. TELEPHONE (Include Area Code) (202) 696-4302 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE