PORTING THE UCSD p-SYSTEM TO UNIX

by

CARLOS LYNN QUALLS

B. S., University of Arkansas, 1975

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1983

Approved by:

Major Professor

TABLE OF CONTENTS

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

ACKNOWLEDGMENTS

I would like to express my appreciation to a loving and understanding wife Jody without whom this project would not have been finished. I would also like to thank my supportive family for their help and encouragement during problem times. I would like to thank Mr. Jim Bandy of SofTech Microsystems for the information he provided allowing this project to start. And to my committee Dr. Virg Wallentine Dept. Head, Dr. Dave Gustafson and Dr. Paul Fisher whose help and support was greatly appreciated.

LIST OF ILLUSTRATIONS

INTRODUCTION

1.1 HISTORY.

The UCSD P-SYSTEM was created by a group of undergraduate and graduate students at the University of California at San Diego (UCSD) under the direction of Kenneth Bowles in late 1974. The computer used by students up until that time was a large and very unfriendly machine running a machine-specific implementation of ALGOL in a batch environment. Access to the machine was therefore limited by the lack of keypunch equipment and the slow turnaround time of the overworked machine.

Dr. Bowles, who was teaching the introductory programming class, wanted to institute changes in the way the programming class was taught. The class would be self-paced to allow for the large number of students and to allow for the different work and study habits of the students. The class would use Pascal instead of ALGOL and microcomputers because of their low cost and ability to allow hands-on interactive experience with the computer. The system was first implemented on PDP 11/10's with floppy disk drives and VT-50 terminals. The students purchased their own floppy disks to hold the system and the programs being worked on by the students.

Even though this was a novel and interesting approach to teaching computing, the university found there was a large commercial interest in a system which could be easily transported to different systems. The system allowed companies to develop a package or program which could run on many systems without changes. The SofTech Microsystems company was formed to support, maintain, license and continue developing UCSD Pascal

and the System that supports it.

1.2 MOTIVATION FOR PROJECT.

The purpose of this project was to install the UCSD P-SYSTEM operating system under the UNIX operating system running on Kansas State University's Perkin Elmer minicomputers. One reason for this was to allow the use of an "industrial standard" version of Pascal, UCSD Pascal, by the students of Kansas State University. Another important reason was to provide a method for allowing students owning their own equipment to do work at home and still have it compatible with a language used by the university for instruction. The initial attempt was to bring up the full UCSD P-SYSTEM operating system (the UCSD P-SYSTEM is described in Chapter 2) . A latter project could be the removing of the UCSD Pascal compiler from the P-SYSTEM and installing only that part under the UNIX operating system. A third project could be a code generator which would take the P-CODE produced by the UCSD Pascal compiler and convert the P-CODE into native code to run on the PERKIN-ELMER minicomputers under the UNIX operating system. A fourth project could be to install the System on the bare Perkin-Elmer (Interdata) 16-bit machines owned (but not used because of lack of a good operating system) by the department of Computer Science.

1.3 ORGANIZATION OF THE REPORT

Chapter 1 of this report is a brief history of the USCD P-SYSTEM and the purpose of this project. Chapter 2 concerns the UCSD P-SYSTEM operating system and some of its commands. Chapter 3 of this report is an explanation of the structure of the P-MACHINE used to run the UCSD

P-SYSTEM. Chapter 4 describes the interperter which was converted as the implementation part of this project. Also discussed are the problems encountered while working on this project. Solutions to some of the problems are brought up and discussed. Chapter 5 discusses what needs to be done to finish the project and where the project should go next.

THE P-SYSTEM

The UCSD P-SYSTEM is an operating system for mini and micro-computer systems. Most of the computers running the UCSD P-SYSTEM are single user micro-computers or single user mini-computers, but there is a new multi-user version out. The P-SYSTEM is different from most micro-computer operating systems (such as CPM) in the fact that only a small portion of the operating system is written in assembly language and the operating system is targeted to many different computers instead of just one type of computer (example: CPM will only run on a 8080 or Z-80 based computer). The interpreter (discussed in a later chapter) or P-MACHINE EMULATOR (PME), as the interpreter is sometimes called, is the only section of the operating system written in assembly language. The rest of the operating system and all support programs are written in UCSD Pascal, which is a slight variation of Standard Pascal[UM] as defined by Niklaus Wirth. To move the operating system from one type of computer to another, the interpreter is the only part which must be rewritten. The P-SYSTEM is designed to run on a system with at least 48K and one floppy disk drive (two floppy disk drives are preferred). The P-SYSTEM contains the operating system, a file handler, a full screen editor, line editor, language compiler (normally Pascal but can be FORTRAN 77 or BASIC), utilitiy programs and manuals.

As the system is distributed normally, the language compiler supplied is UCSD Pascal. For those who do not like the Pascal language or those who decline to learn another language, the system can be bought with either FORTRAN 77 or a BASIC complier. The system is currently implemented on the Intel 8080/8085, Intel 8086, Intel 8088, MOS

Technology 6502, Motorola 6800, Motorola 68000, IBM PC, APPLE computers, National Semiconductor NS16000, Western Digital Micro-engine and others.

The UCSD P-SYSTEM operating system was designed to be a very friendly system to the inexperienced user, but it was also designed to be efficient for use by the experienced user. In order to accomplish both of these goals, the system was designed as a "menu-driven" system. This funtion almost always displays a prompt line at the top of each program screen. The prompt line concept allows the experienced user to do what he needs to do without slowing him or her down, but it also gives the inexperienced user more information to help him or her do what he needs to do. The prompt lines are of the following format:

NAME OF PROGRAM: C(ommands [version number].

The C(ommand format is used to show that the command is executed by typing the letter before the "(". The command letter is also capitalized. The part of the command after the "(" is used to make the command self-explanatory. Example: X(ecute is the command to execute a program and the command letter is X. In relation to most commands in the system, there are two special files which most of the commands take as being the default files if no other file name is given. These files are called SYSTEM.WRK.TEXT and SYSTEM.WRK.CODE. The file SYSTEM.WRK.TEXT contains the source code for the file which is currently being worked on. If there is no current work file, the commands (after looking for the default filename) ask for the name of the file to work with and names a copy of that file SYSTEM.WRK.TEXT. The file SYSTEM.WRK.CODE is the file which contains the object code (after running the source code through either a complier or an assembler) of the file currently being worked on. If the file currently being worked on is a new file or one which has

never been complied or assembled, the file SYSTEM.WRK.CODE is probably undefined or non-existent.

The system commands form a hierarchical tree structure (see Figure 2.1) with the first level of the tree consisting of the following commands:

```
A(ssem
C(omp
E(dit
F(ile
H(alt
I(nit
L(ink
 M(on
R(un
U(ser
X(ecute
   ?
```

The A(ssem command is the command to run the system assembler called SYSTEM.ASSMBLER using the file SYSTEM.WRK.TEXT. This file contains the source code to be assembled. If the default file does not exist, the assembler prompts the user for the name of the source file. The assembler also prompts the user for the name of the file into which the object code is to be placed and the name of a file where a source listing is to be placed. The defaults for these are SYSTEM.WRK.CODE for the object code and no listing generated for the listing file. The assembler can only produce native code for one machine. The code produced depends on which assembler is in the file called SYSTEM.ASSMBLER. This could be either a 8080, Z80, 6502 or whatever assembler the user happens to be using.

The C(omp command causes the program called SYSTEM.COMPILER to be run with the default file SYSTEM.WRK.TEXT if it exists. If the default

file does not exist, the compiler prompts the user for a file name to compile. The compiler produces P-CODE which is either placed in the file called SYSTEM.WRK.CODE or in a file specified by the user. If there is a error in the source file, the compiler gives the following error message:

LINE ##, error ###: <sp>(continue), <esc>(terminate), E(dit.

The user then has the choice of typing a space to continue the compiler, using the escape key to stop the complier, or utilizing the "E" key to stop the complier and go into the editor. If the user chooses to go into the editor, the editor brings up the source file and positions the cursor on the line where the complier found the error. This allows the user to quickly make necessary changes and run the complier again. The user could instead continue the complier and get a listing of all the errors before going into the editor and making changes in the source file.

The E(dit command executes the file called SYSTEM.EDITOR. This can be one of two different types of editors. The normal (the way the system comes) editor is a full screen editor to be used with a CRT terminal. The other editor is called YALOE which stands for Yet Another Line Oriented Editor. This editor is used by users who either do not like full screen editors or by users who do not have a CRT terminal but instead have a line oriented console such as a DECWRITER. Again the default file is SYSTEM.WRK.TEXT. If it does not exist or is the wrong file, the editor will prompt the user for the name of the file to edit.

The F(ile command executes the program called SYSTEM.FILER. This program allows the user to move files, copy files, rename files, and change the system work files. If the reader wants to find out more

information about what and how the filer works, I refer him or her to the USERS MANUAL put out by SofTech Microsystems or to any of a number of books currently available on the UCSD P-SYSTEM.

The H(alt commands stops the execution of the system. On some implementations, this causes the system to re-bootstrap itself, but on most systems the user has to bootstrap the system back up.

The command I(nit executes the program called SYSTEM.STARTUP. This program is executed only if it exists. It is also executed (again if it exists) after the system is bootstraped up. The file SYSTEM.STARTUP can be used by the user to cause different things to happen when the system is brought up. The user may want his or her system to be a turnkey operation. The program SYSTEM.STARTUP will allow this type of operation. "Non-fatal" runtime errors cause the system to execute an initialize command automatically.

The command L(ink executes the program called SYSTEM.LINKER which allows the user to link together native code routines created by the assembler with those generated by the complier. This might be necessary in cases where very time critical routines have been written in assembly language instead of Pascal to increase the execution speed of the time critical sections.

The command M(on causes all the command which follow to be stored in a file. This allows the user to remember or save a sequence of commands. This sequence of commands might be used later as the command sequence for the file called SYSTEM.STARTUP (see the I(nit command). The sequence of commands also could be used by the editor to make the same

changes to several files without the user worrying about making mistakes retyping the commands each time.

The R(un command causes the file called SYSTEM.WRK.CODE to be executed if it exists. If it does not exist, then the user is prompted for the name of the code file to be executed. If all code which is necessary to execute the code file does not exist, the linkler is automatically called and the system library is searched for the necessary code to be linked in. If the code is not in the system library, an error occurs.

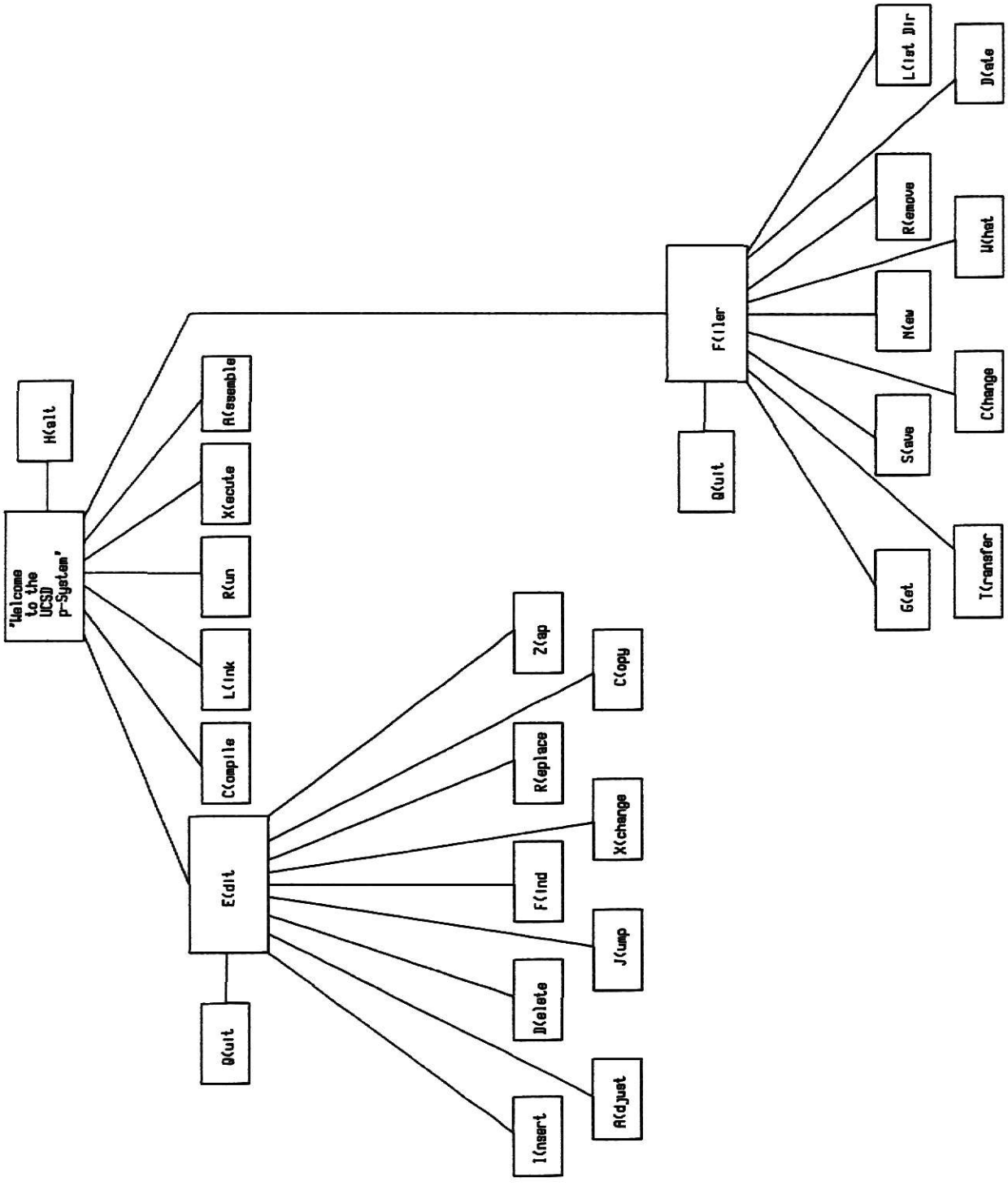
The command U(ser caused the last program executed to be executed over again. It will not restart the compiler or the assembler.

The X(ecute command prompts the user for the name of the code file to be executed. If the file cannot be found or if all the necessary code has not been linked into the file, an error message is returned to the user explaining the problem.

The ? command is used by the user to view other commands which my not appear on the first prompt line. One reason some of the commands might not appear on the prompt line is that several of the systems currently running the P-SYSTEM do not have a 80 column screen but instead have a 32, 40 or 64 column screen. The system is set up with the size of the screen when it is first brought up on a particular piece of hardware. The system then remembers the screen size each time the system is rebooted and only the commands which fit on one line of the screen are displayed.

The commands presented above are the first level of commands which are available to the user. Other commands exist under the filer and

under the editor. If the reader wants more information about the other commands, I refer the reader to the USERS MANUAL by SofTech Microsystems or to any of a number of books currently out on the UCSD P-SYSTEM.



COMMAND TREE - ONLY THE MOST FREQUENTLY USED COMMANDS
Figure 2.1

THE P-MACHINE

The P-MACHINE is an idealized stack oriented machine. The code native to the P-MACHINE is called P-CODE. The "P" in both P-MACHINE and P-CODE stands for pseudo. P-CODE was designed to be easy for a compiler to generate. It was also designed to be very compact so that a program written in P-CODE is usually smaller than the equivalent program written in the native code of a "real" processor. The P-MACHINE has been realized as a physical processor (the WESTERN DIGITAL MICRO-ENGINE), but it is usually emulated on existing processors (Z-80's, 6502's, 68000's, etc.). The emulation is done by a program called the interpreter which decodes and executes P-CODE on the real processor. The interpreter also provides an interface between the P-MACHINE and the real world. Thus, the interpreter handles all of the input/output and interaction with external devices required by the P-MACHINE.

P-CODE can be considered to be the native (natural) language of the P-MACHINE, just as one would call assembly language the native language of a particular real processor. The P-MACHINE has several different registers associated with its operation (see below). Unlike other processors, the registers are reserved for specific duties and are not usually affected by the P-CODE instructions. Instead, the P-CODE instructions use an intergal part of the P-MACHINE called the stack for temporary data storage and other tasks normally handled by the general purpose registers of most processors. Most of the P-CODEs affect the stack either directly or indirectly. The P-CODE instructions are broken down into the follow types:

Constant One-Word Loads

Local One-Word Loads and Stores
Global One-Word Loads and Stores
Intermediate One-Word Loads and Stores
Extended One-Word Loads and Stores
Indirect One-Word Loads and Stores
Multiple-Word Loads and Stores
Byte Load and Store
Packed Field Load and Store
Record and Array Indexing and Assignment
Logical Operators
Integer Arithmetic
Real Arithmetic
Set Operations
Byte Array Comparisons
Jumps
Routine Calls and Returns
Concurrency Support
String Instructions
and Miscellaneous Instructions.

The break down of the P-CODE instructions is very similiar to the break down of assembly language for most processors, with the exception of String Instructions, Set Operations and Concurrency Support. Some of the registers of the P-Machine are

SP points to the word that is on the top of the P-MACHINE stack

IPC points to the next P-CODE to be executed

CURPROC Contains the procedure number of the currently executing procedure

CURTASK points to the Task Information Block (TIB) of the currently executing task

EREC points to the Current Environment Record

MP points to the current activation record

READYQ points to the TIB at the head of the queue of tasks ready to run.

Figure 3.3 shows the hierarchy layout of the P-SYSTEM. The

application programs or system programs make calls to the operating system. If the program is requesting input or output then the operating system makes a call on behalf of the program to either the file or screen I/O routines. The operating system and the file and screen I/O routines are written in UCSD Pascal and compiled down to P-CODE. The P-CODE is executed on a host processor by the P-MACHINE EMULATOR (PME) or interpreter. Any calls to I/O routines are routed by the interpreter to a section of code in the interpreter called the Basic Input Output Subsystem (BIOS). The BIOS routines then make calls on user written interfaces to the physical host processor and peripherals. In this project the BIOS makes calls on the UNIX operating system. UNIX then performs the requested action and returns a completion code.

The P-SYSTEM is a collection of UCSD Pascal programs. Some of these programs are intended to be used by programs other than the operating system. The name of the program reflects the function of the program. The operating system programs are as follows[IAG]:

HEAPOPS EXTRAHEAP PERMHEAP	Heap operations
SCREENOPS	Screen control
FILEOPS	File and Directory operations
PASCALIO EXTRAIO SOFTOPS	File-level I/O
SMALLCOMMAND COMMANDIO	I/O redirection and chaining
STRINGOPS	String intrinsics
OSUTIL	Conversion utilities
CONCURRENCY	Concurrency

REALOPS	Real Number I/O
LONGOPS	Long Integer operations
GOTOXY	Screen cursor control
KERNEL	Nonswappable central facilities of operating system (always resident in main memory).

Page 112 of the SofTech Microsystem Internal Architecture Guide says the following about the KERNEL.

"KERNEL contains the resident code necessary to maintain the codepool, handle faults, and read segments. The Kernel also contains four subsidiary segments, which are swappable:

GETCMD processes user input at the main command level, and builds a user program's runtime environment;

USERPROG is the reserved segment slot for the user's program (at bootstrap time it contains the Pascal-level code which builds the initial runtime environment for the Operating System);

INITIALIZE is called when the System is booted or re-initialized: it reads SYSTEM.MISCINFO, locates the System codefiles, and sets up the table of devices;

PRINTERROR prints runtime error messages.

The Operating System UNITs are compiled separately. They are bound together in a single codefile, SYSTEM.PASCAL, by using the utility LIBRARY.

Because of certain bootstrap restrictions, KERNEL must always reside in segment-slot 0 and USERPROG must always reside in slot 15. There are no other restrictions on the location of units within SYSTEM.PASCAL."

Two dynamic structures called the Stack and the Heap are maintained by the system for memory-resident data. The Stack is used for static variables, bookkeeping information about procedure and function calls, and evaluation of expressions. The Heap is used for dynamic variables, including the structures that describe a program's environment[IAG]. While the Heap is an intergal part of the system, it is primarily supported by the operating system and not by the P-MACHINE. The Stack and

the Heap reside in main memory at opposite ends and grow toward each other in a first-in-first-out fashion. The area in memory between the Stack and the Heap is occupied by the Codepool. The Codepool is the section of memory where programs or code segments are loaded to execute.

Figure 3.1[IAG] shows a snapshot of main memory while the system is running. A subset of the operating system (the KERNEL) is always resident in the system. The interpreter is also always resident in the system usually at low memory. A program, if it contains SEGMENT routines or EXTERNAL native code routines, will have to have access to more than one code segment as it is running in the system. Both the calling and called segment must be in main memory for an inter-segment call to succeed. When a program is run, the operating system reads reference information about each segment and builds tables to be used at runtime for references from one segment to another.

Listing 3.3[IAG] shows the structure of the first block of a codefile which is the segment dictionary for that file. If the dictionary is longer than one block, the other dictionary entries are imbedded in the codefile in a linked list arrangement. As shown in listing 3.3, the information describing each segment of a codefile is contained in 6 different arrays. Each segment dictionary entry can therefore handle 16 different segments.

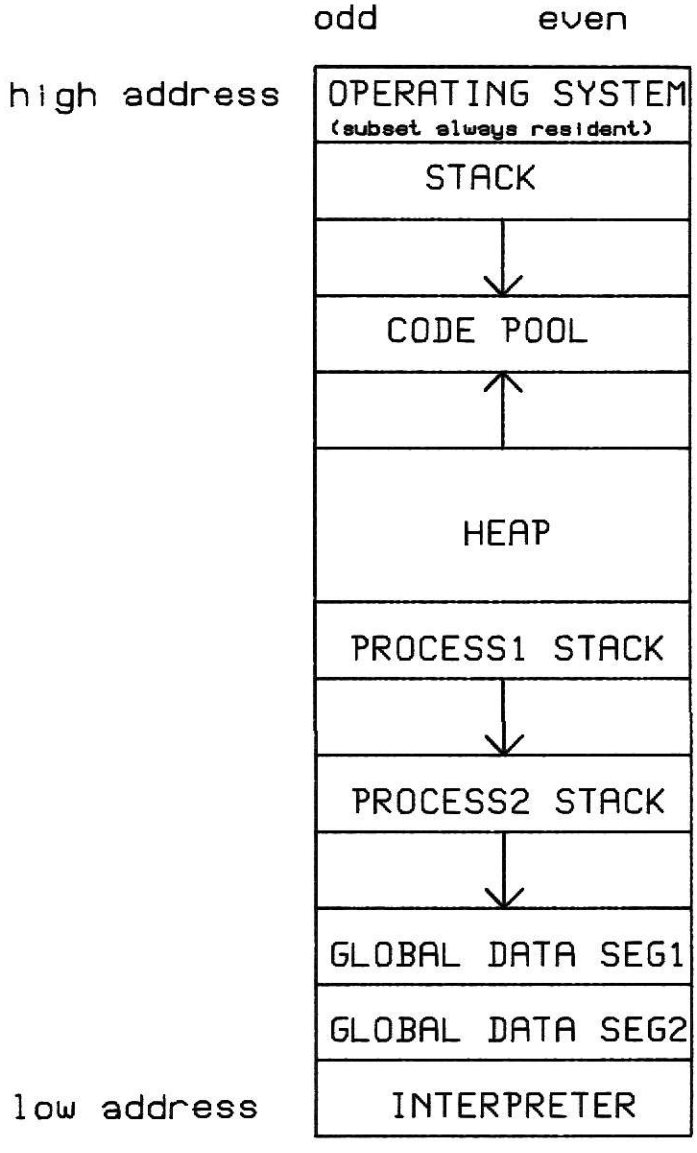
Figure 3.2[IAG] shows the layout of individual segments within the codefile. Code segments are a collection of routines and descriptive information to be used by the operating system. The unit of movement for code in the operating system is a segment; therefore a segment must be a contiguous area both on the disk and in memory. Code is loaded into

memory a segment at a time. If the operating system runs out of memory, unreferenced segments are swapped out of memory to make room for currently referenced segments. The procedure dictionary points to the routine dictionary. The routine's number is an index into the routine dictionary with the n'th word containing a pointer to routine n. There is from 1 to 255 different routines that a segment can contain.

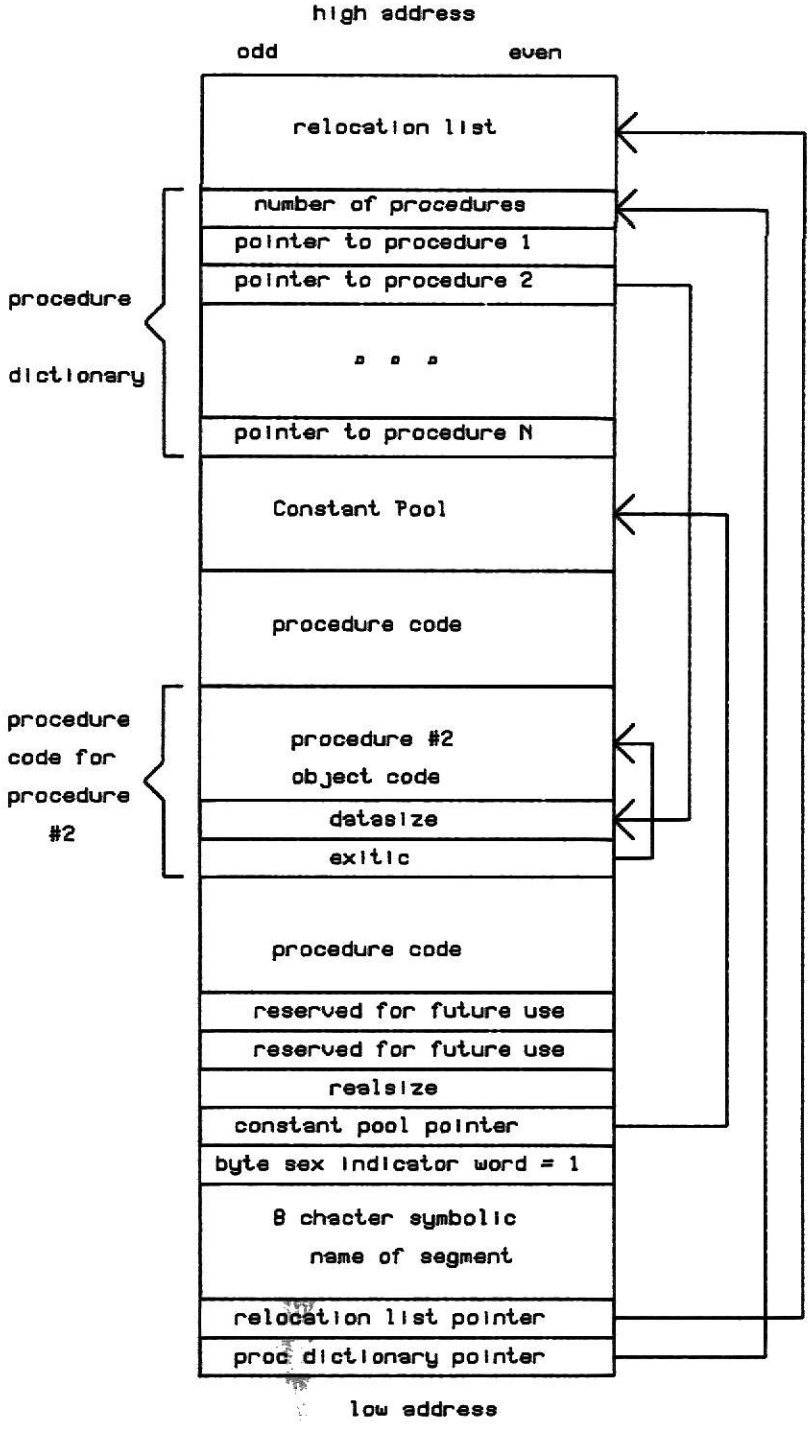
A Segment Information Block (SIB) is allocated on the Heap by the operation system for each active code segment. The SIB remains on the Heap for the entire time that a segment is active even though the segment may not be in memory but on disk. Listing 3.4[IAG] shows a Pascal code fragment which describes the SIB. An SIB is active (may be used by a program) as long as the Link_Count field is greater than 0. When the Link-Count field reaches 0, the SIB is removed from the Heap because the programs which were using the segment have finished with the segment.

At any one time, the P-MACHINE may have at most one task running, several tasks ready to run and several tasks waiting on semaphores. Tasks ready to run and tasks waiting on semaphores are organized into two different queues. Within each queue, the tasks are ordered according to their priority as assigned by the system. When the system first starts up and when there is no other task running then the default task (called the "main task") is run. This task is a section of the operating system itself which displays the main prompt line and waits for the user to descide what he or she wants to do. Other tasks, such as the editor and the file handler can be started by the user at almost any time. These tasks consist of three things: the task body, the Task Information Block (TIB, see listing 3.5), and the task stack. The TIB

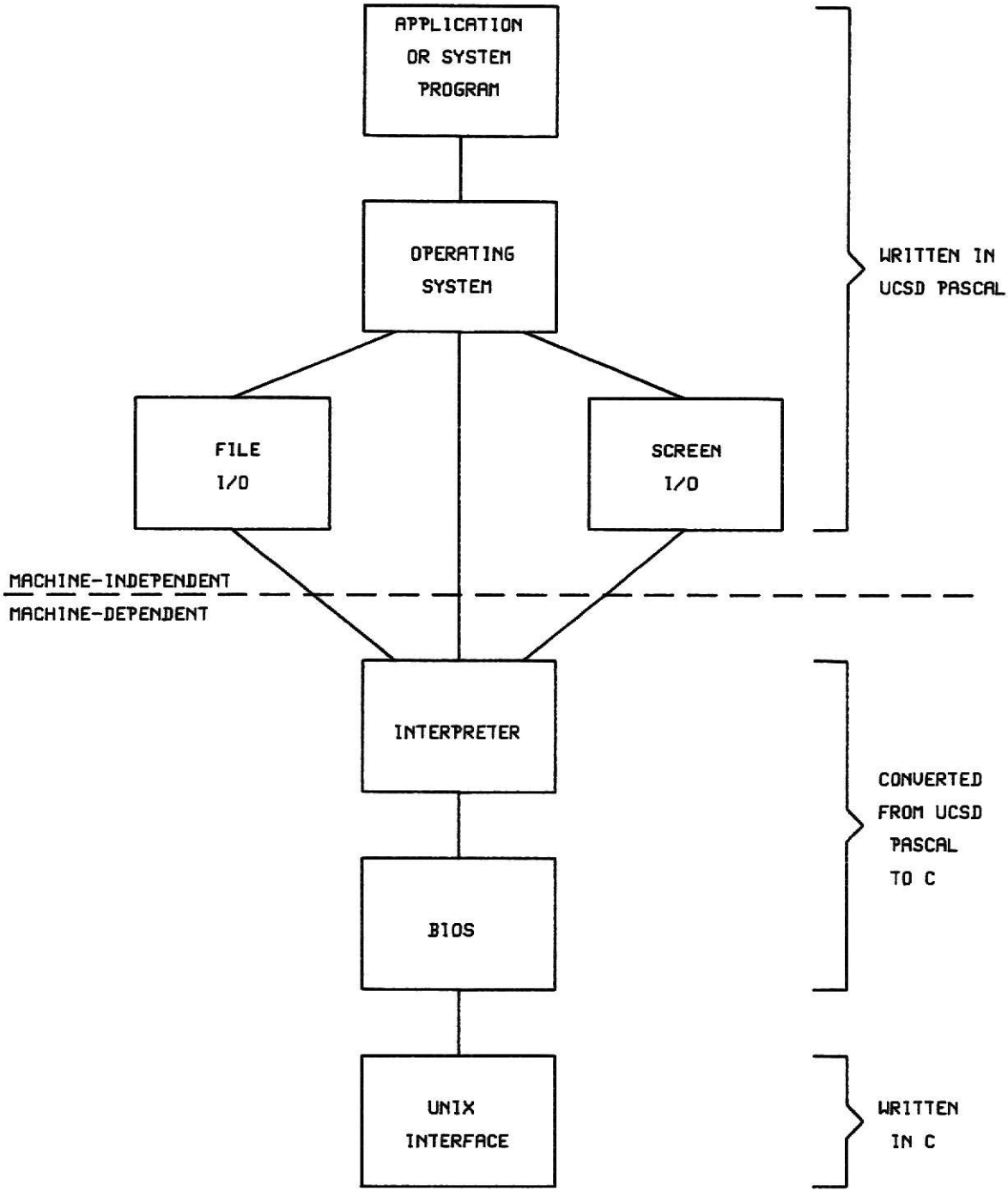
contains information about the task's execution environment, which must be maintained and restored if a task is restarted after being idle for any reason. Each task, except the main task, has a seperate task stack (refer back to figure 3.1) allocated on the Heap. One thing contained there is the task activation record. The size of the task stack is normally 200 words unless it is changed with the STACKSIZE parameter of the START intrinsic.



MAIN MEMORY USAGE
Figure 3.1



EXECUTABLE CODE SEGMENT FORMAT
Figure 3.2



P-MACHINE HIERARCHY
Figure 3.3

```
CONST Max_Dic_Seg = 15;{maximum segment dictionary record entry}

TYPE Seg_Dic_Range = 0..Max_Dic_Seg;{range for segment dictionary entries}

     Segment_Name = PACKED ARRAY [0..7] OF CHAR;{segment name}

     {segment types}
     Seg_Types = (No_Seg,      {empty dictionary entry}
                  Prog_Seg,    {program outer segment}
                  Unit_Seg,    {unit outer segment}
                  Proc_Seg,    {segment procedure inside program or unit}
                  Seprt_Seg); {native code segment}

     {machine types}
     M_Types = (M_Psuedo, M_6809, M_PDP_11, M_8080, M_Z_80,
                M_GA_440, M_6502, M_6800, M_9900, M_8086,
                M_Z8000, M_68000);

     {p-machine versions}
     Versions = (Unknown, II, II_I, III, IV, V, VI, VII);

     {segment dictionary record}
     Seg_Dict = RECORD
       Disk_Info:
         ARRAY [Seg_Dic_Range] OF {disk info entries}
         RECORD
           Code_Addr: integer;  {segment starting block}
           Code_Leng: integer;  {number of words in segment}
         END{of RECORD};
       Seg_Name:
         ARRAY [Seg_Dic_Range] OF Segment_Name;{segment name entries}
       Seg_Misc:
         ARRAY [Seg_Dic_Range] OF {misc entries}
         PACKED RECORD
           Seg_Type: Seg_Types;     {segment type}
           Filler: 0..31;           {reserved for future use}
           Has_Link_Info: Boolean; {need to be linked?}
           Relocatable: Boolean;    {segment relocatable?}
         END{of PACKED RECORD};
       Seg_Text:
         ARRAY [Seg_Dic_Range] OF integer;{start blk of interface text}
       Seg_Info:
         ARRAY [Seg_Dic_Range] OF {segment information entries}
         PACKED RECORD
           Seg_Num: 0..255;          {local segment number}
           M_Type: M_Types;          {machine type}
           Filler: 0..1;             {reserved for future use}
           Major_Version: Versions; {P-machine version}
         END{of PACKED RECORD};
```

PASCAL CODE FRAGMENT FOR SEGMENT DICTIONARY
Listing 3.3

```
  Seg_Famly:
    ARRAY [Seg_Dic_Range] OF {segment family entries}
    RECORD
      CASE Seg_Types OF
        Unit_Seg, Prog_Seg:
          (Data_Size: integer;        {data size}
           Seg_Refs: integer;         {segments in compilation unit}
           Max_Seg_Num: integer;      {number of segments in file}
           Text_Size: integer);       {# of blks interface text}
        Seprt_Seg, Proc_Seg:
          (Prog_Name: Segment_Name); {outer program/unit name}
    END {of Seg_Famly};
  Next_Dict: integer; {block number of next dictionary record}
  Filler: ARRAY [0..6] OF integer; {reserved for future use}
  Copy_Note: string[77]; {copyright notice}
  Sex: integer; {machine sex (Sex = 1)}
END {of SEC_DICT};
```

PASCAL CODE FRAGMENT FOR SEGMENT DICTIONARY
Listing 3.3

```
SIB = RECORD
        Seg_Base: Mem_Ptr;      {segment's memory location}
        Ref_Count: integer;     {# of active calls to the seg}
        Activity: integer;      {memory swap activity}
        Link_Count: integer;    {number of links to the SIB}
        Residency: -1..maxint; {-1 = pos lock, 0 = swap, n = mem lock}
        Seg_Name: PACKED ARRAY [0..7] OF CHAR;
        Seg_Leng: integer;  {# of words in segment}
        Seg_Addr: integer;  {disk address of segment}
        Vol_Info: Vl_Ptr;   {pointer to disk drive info}
        Data_Size: integer; {number of words in data segment}
        Res_SIBs: RECORD    {code pool management record}
                    Next_SIB: SIB_P;              {next SIB in list}
                    Prev_SIB: SIB_P;              {previous SIB in list}
                    CASE Boolean OF               {scratch area}
                      TRUE: (Sort_SIB: SIB_P);    {next SIB in sort list}
                      FALSE: ( New_Loc: Mem_Ptr); {temporary address}
                  END {of Res_SIBs};
      END {of SIB};
```

PASCAL CODE FRAGMENT OF SEGMENT INFORMATION BLOCK
Listing 3.4

```
TIB = RECORD {Task Information Block}
        Regs: PACKED RECORD
                Wait_Q: TIB_Ptr;
                Prior: byte;
                Flags: byte;
                SP_Low: Mem_Ptr;
                SP_Upr: Mem_Ptr;
                MP: MSCW_Ptr;
                BP: MSCW_Ptr;
                IPC: integer;
                Env: ERec_Ptr;
                ProcNum: byte;
                TIBIOResult: byte;
                Hang_Ptr: Sem_Ptr;
                M_Depends: integer;
              END {of Regs};
        MainTask: boolean;
        Start_MSCW: MSCW_Ptr;
      END {of TIB};
```

PASCAL CODE FRAGMENT OF TASK INFORMATION BLOCK
Listing 3.5

INTERPRETER

This project was made possible by the efforts of Jim Bandy, Applications Development Manager at SofTech Microsystems. Mr Bandy allowed access to a copy of the UCSD P-SYSTEM INTERPRETER VERSION 4 [v4int] written by Michael Harrison at IMOS in England. This Pascal realization of the VERSION 4 INTERPRETER is the basis for this project. Mr. Bandy also provided the following for the project.

The UCSD P-SYSTEM Installation Guide,
The UCSD P-SYSTEM Internal Architecture Guide,
The UCSD Pascal Users Manual,
The UCSD Users Manual Supplement,
BEGINNER'S GUIDE FOR THE UCSD PASCAL SYSTEM,
by Kenneth L. Bowles
and
The UCSD Adaptable P-SYSTEM on 8" floppy disk.

The Pascal realization was converted into C as the first step of the project. The listing of the C version is in the APPENDIX. The conversion was not as easy as one might first assume. Several functions of Pascal had to be "faked" under the C language. This kind of experience helps one to appreciate differences between languages and implementations of those languages. The conversion process kept to the basics of the C language and did not use any new extensions or non-portable functions. This was to allow the C version to be ported to any machine having a C complier.

The UNITREAD statements where converted to a lseek statement followed by a read statement. This combination proved to be a fairly close representation of the UNITREAD statement.

Several of the data structures in the Pascal realization were declared as variant records allowing those structures to be accessed at

different times as different length values. This type structure was represented in the C version as a union. The union is a special case of the structure declaration in the C language. By addressing fields in the structure with different names at different times different sizes can be applied to the structure.

EXAMPLE:
```
union example
{
   short  u_integer;
   char   u_char[2];
}
```

If one assigns a value to the union by typing ex.u_integer = 25; then one is dealing with a 16-bit value. If one accesses the union by using ex.u_char[0] or ex.u_char[1] then one is dealing with an 8-bit value. The reason C does not complain about this is because C makes little distinction between a character and an integer. A character can be promoted to an integer and an integer can be converted to a character by the assignment operator.

One of the hardest problems caused by the conversion process was how to "fake" sets in C without resorting to a language extension. The answer to this is really quite simple if the size of the set is 64-bits or less and if one remembers that sets deal with an enumeration type. The following is an example of how sets can be implemented or "faked" in C [RY83]. The bit shifting operators << and >> are used to set and unset bits in a variable. The bits in a set variable are used to keep track of the members of the set. Since sets are a enumeration type, bit 1 is used for the first member of the set, bit 2 for the second member etc.. For the following example A and B are declared to unsigned short integers. This would allow for a set of size 16. To get a larger set one could use unsigned integer for a 32 bit set or unsigned double for a 64 bit set

size.

```
For example, if you number the bits from right to left,
then A |= 1<<B is equivalent to A := A + [B]
     A &= ~(1<<B) is equivalent to A := A - [B]
and  if (A & (1<<B)) is equivalent to IF (B IN A)
```

Everything up to this point was fairly straight forward. The conversion took about 80 to 100 hours and was accomplished with no software tools except a full screen editor and a lot of head-scratching. The real problems still laid ahead. The question "Why does the program not function?" has be posed by every programmer who has ever written or converted a program consisting of more than 500 or so lines of code. The interpreter in Pascal consisted of over 3000 lines of code. After the conversion to C the interpreter was a little over 4000 lines of code. The reader might ask well ask why. C is suppose to do more in less lines of code than Pascal. One reason for the great expansion in code is C's lack of enumeration type or for that matter C's almost total lack of types and the lack of C's ability to define new types (a typedef statement is available and used, but it does not allow the programmer to define new types in the same way Pascal does). All enumerations in the Pascal program had to be separated into individual lines of the form #define <enumeration name> <value>;.

The project now slowed down as all of the material sent had to be read and re-read to try and understand the inner workings of the UCSD P-SYSTEM. A means of moving the information from the 8" floppy disk to the UNIX system also had to be found. The transfer was accomplished with the help of Mike Schwarz and the Agricultural Engineering PDP-11 computer. The Engineering computer runs a UNIX Version 6 operating system and has 8" floppy disk drives as part of its hardware. The floppy disks

were transferred to tape in the UNIX "tar" format and then downloaded onto the UNIX system in the Computer Science Department's mini-lab.

The first real break came when it was discovered that the P-SYSTEM on the floppy disk was byte-swapped. If one refers back to Listing 3.3, there is a field called sex of type integer. This field is the sex of the machine which compiled and created the disk copy. The sex field is a 16-bit field with a integer 1 stored in it. On machines of the same sex, when the operating system checks this field the operating system will see an integer 1. On machines of different sex, the operating system will see an integer 256 indicating the disk is of the wrong sex. The Internal Architecture Guide had this to say about the byte sex:

"There are two groups of word-oriented (byte-sex-dependent) information. The first is the superstructure, such as the routine dictionary. This information is flipped by the Operating System when a segment is loaded. The second is embedded information, such as for example, constants (accessed by LDC) or XJP tables. This sort of information is flipped by the Interpreter."[IAG]

The Pascal realization of the P-SYSTEM Interpreter makes little mention of the byte-sex and in most cases ignores it all together. The first change made to the interpreter was in the bootstrap section of code. On the adaptable system the disk is not laid out as one might expect (see Figure 4.1). The first whole track (track 0) and the first 1024 bytes of the second track (track 1) are either not used or contain information not needed by this project. The UNITREAD in the Pascal version of the interpreter was replaced with an lseek which skipped over the first 4352 bytes to find the disk directory. Figure 4.2 contains the layout of the disk directories. The interpreter was changed so that it checked the second field of the disk directory. If the value there was 255 or less then the disk directory is not byte-swapped. If the value there is 256

or over the the directory is byte-swapped and word-oriented (16-bit) information must be byte-swapped by the interpreter before it is used. This was done with the following C code.

```
/*
   This structure defines a section of memory which can be
   accessed as either a 16-bit integer or 2 8-bit characters
*/
union swapper
{
     short sint;
     char  schar[2];
} swap1,swap2,swap3;



/*
   The following code gets a 16-bit value into a temporary location
   and then swaps the value using another temporary location.
   The value in the second temporary location is the byte-swapped
   value to be used in following calculations
*/
         swap1.sint = /*The byte-swapped value*/
         /* swap the values */
         swap2.schar[0] = swap1.schar[1];
         swap2.schar[1] = swap1.schar[0];
         /* variable */ = swap2.sint;
```

The following section of code shows the structure of a section of memory called store. If the above quote from the Internal Architecture Guide is to be taken for face value, then every place in the interpreter where the structure store is accessed as either the field data or the field fset (see structure below) the value obtained must be checked two ways before being used. One check must be to see what segment the address being accessed is in. This can be done by checking the segment base field and segment word field of the Segment Information Block (refer back to Listing 3.4). The other check must see if the segment we are accessing is a byte-swapped segment or not. This can be done by checking the sex field of the segment dictionary (refer back to Listing 3.3) for the segment. If the memory location fits both of the above checks then

it is byte-swapped and the interpreter must use the above byte-swapping technique before any information is used or stored in memory. The Pascal realization of the interpreter makes no mention of either the above checks.

```
/*
   The following section of code is used to describe (fake)
   a section of main memory in a micro-processor.
   The size of the memory section is 16k words or 32k bytes.
   This size is considered to be the minimum contiguous memory
   size needed for the UCSD P-SYSTEM. The P-SYSTEM needs a
   minimum of 45k bytes, not all of it needs to be contiguous.
*/

/* ********************************************************************** */
/* the following data structure defines p_machine store - store has a     */
/* variety of types imposed on it.  For this reason the data structure    */
/* is punned to permit access of 16 bit integers-INTEGER; bytes, 8 bit    */
/* characters-fchar, operator codes-func and 16 bit sets-fset             */
/* ********************************************************************** */
   union store_structure
   {
       short           data  [16000];
       char            code  [32000];
       char            fchar [32000];
       char            func  [32000];
       unsigned short  fset  [16000];
   }store;
```

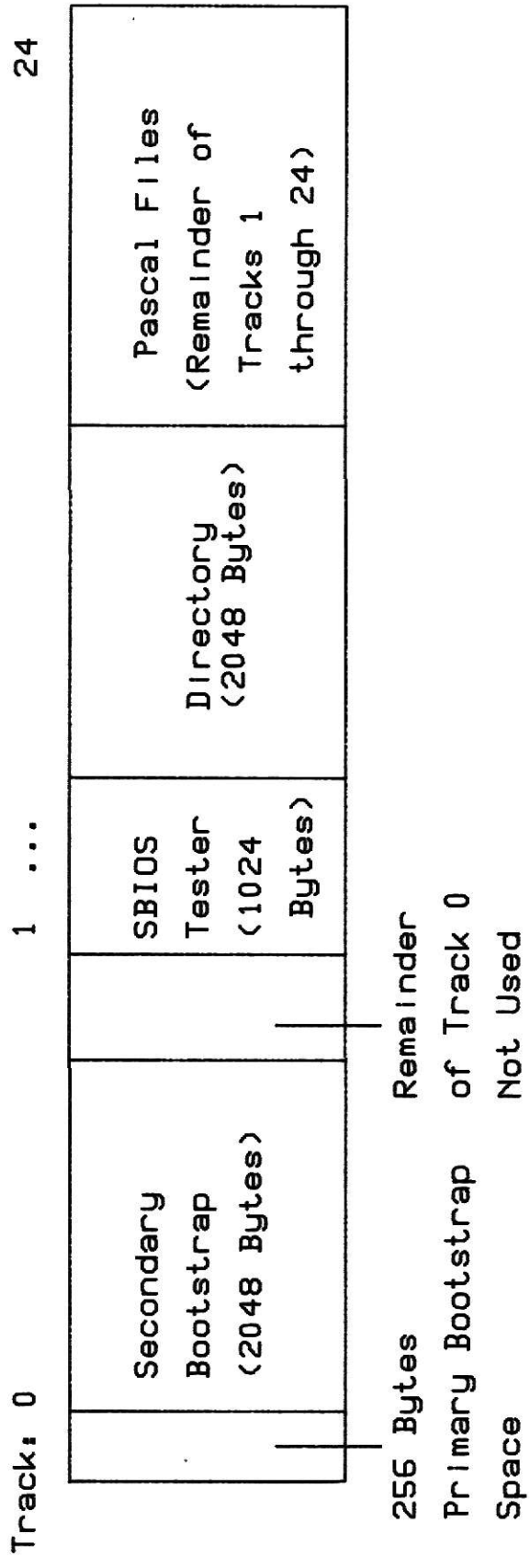
After the above problem with byte-swapping was solved, by inserting the needed checks, the next problem emerged. This problem is related to a section of the interpreter which made a call to a routine called sp_rtns. The routine sp_rtns (special routines) did not exist in the first copy of the Pascal realization of the interpreter. The sp_rtns routine is the section of the interpreter which supports the RUNTIME SUPPORT PACKAGE(RSP)[TP83]. The RSP is the machine independent interface section to the machine dependent BASIC INPUT OUTPUT SUBSYSTEM(BIOS) of the P-SYSTEM. If a copy of the interpreter exists for a specific computer type then the only section of the P-SYSTEM which must be rewritten is the boot-strap routine and the BIOS. This is one of the features

which makes the P-SYSTEM easy to port from one machine configuration to another. The BIOS is a very simple interface between the interpreter and the computer, which even a inexperienced programmer should be able to code.

After the code for the RSP arrived, it was converted from Pascal to C and installed in the interpreter. A new problem immediately emerged. The UCSD P-SYSTEM uses unit numbers as shown in the following table.

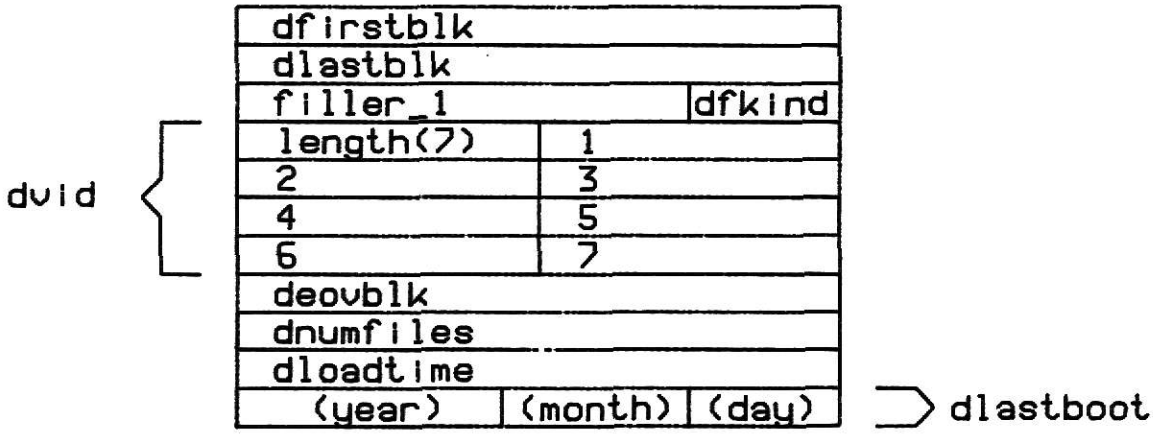
Unitnumber	Volume Name
0	<reserved for the system>
1	CONSOLE
2	SYSTERM
3	<reserved for the system>
4	disk 0
5	disk 1
6	PRINTER
7	REMIN
8	REMOUT
9	disk 2
10	disk 3
11	disk 4
12	disk 5
13-127	<reserved for future expansion>

While C uses unit number 0 for terminal input (stdin), number 1 for terminal output (stdout) and 2 for error messages to the terminal (stderr). The stdin and stdout can also be redirected to files and other devices under the UNIX operating system. When a new file is opened by a C program under the UNIX operating system, then a file descriptor (a number between 3 and the maximum number of files available to that program) is returned and used by the C program. Therefore, a switch statement (Pascal case statement) has to be installed in all routines which interface to the outside world. The switch statement intercepts the call and decodes it to the correct unit number. The switch statement will also connect and disconnect files associated with or asked for by the program.

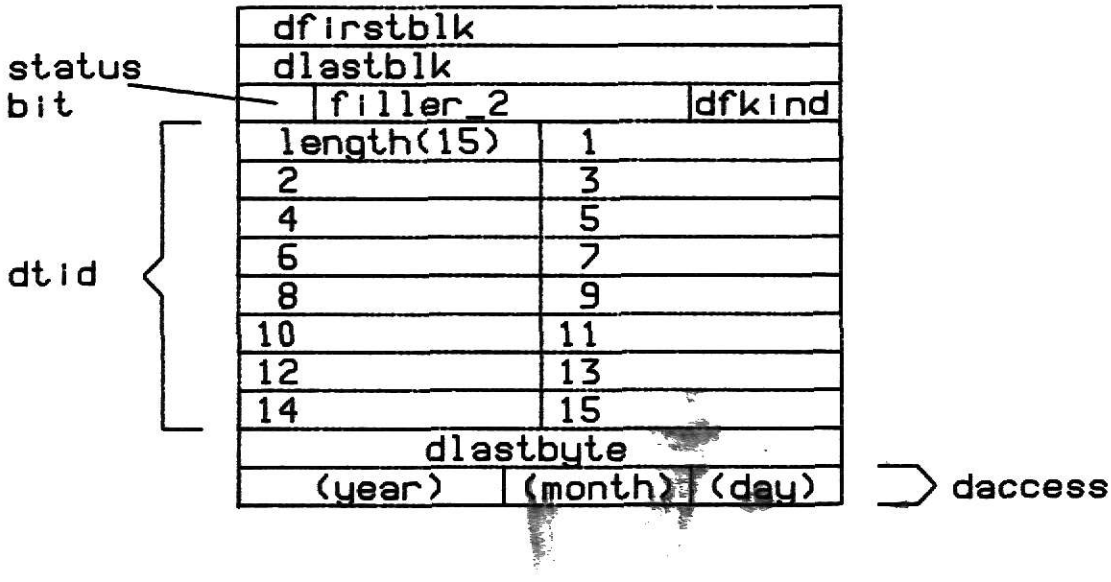


LAYOUT OF AN ADAPTABLE SYSTEM LOGICAL DISK
Figure 4.1

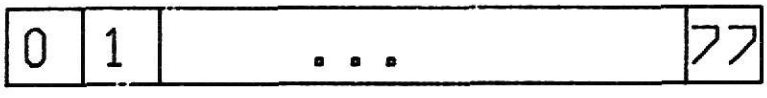
DIRENTRY RECORD (0)
for dfkind=securedir,
untyped file (dir[0])



DIRENTRY RECORD (1-77)



DIRECTORY: array [0..77] of direntry;



DISK DIRECTORY FORMAT
Figure 4.2

RECOMMENDATIONS

After the interpreter is fully tested and installed under the UNIX operating system, the next logical step would be to remove the UCSD Pascal complier from the P-SYSTEM. The complier could then be bought up without the P-SYSTEM operating system as a separate Pascal complier. This would allow students to create Pascal programs using the full screen editor under UNIX. In it's present state the student must create programs under the P-SYSTEM full screen editor and file the programs on psudo floppy disk drives maintained by the P-SYSTEM on the UNIX hard disk drives.

The next logical step would be to take the P-CODE generated by the UCSD Pascal compiler and translate that into native code for the UNIX system. This would probable be Perkin-Elmer machine language. It is questionable at this point as to whether this conversion would speed up program execution.

An offshoot of this project would be to convert the interpreter into Perkin-Elmer assembly language. The interpreter could then be installed on one of Kansas State University Computer Science Departments Perkin-Elmer model 7/16 mini-computer. This would be very comparable to the orginial installation of the P-SYSTEM designed for the PDP-11/10s. This step could put currently unused equipment into useful service. Allowing students hands on experience with a different type of computer, and experience with the type of operating system currently available for personal (home) computers.

REFERENCES

IG The SofTech Microsystems Company. UCSD p-System Installation Guide, 1981.

IAG The SofTech Microsystems Company. UCSD p-System Internal Architecture Guide, 1981.

UM The SofTech Microsystems Company. UCSD Pascal Users Manual Version IV.0, 1981.

UMS The SofTech Microsystems Company. UCSD Pascal Users Manual Supplement Version IV, 1981.

RY83 Private conversations with Robert Young, KSU Instructor, 1983.

TP83 Private conversation with Ted Pal, Consultant, 1983.

APPENDIX A

DIFFERENCES BETWEEN STANDARD PASCAL AND UCSD PASCAL[UM]

String Handling.

STRING is a new data type in UCSD Pascal which of a PACKED ARRAY OF CHAR with a length. Strings may be assigned, passed, and input or output. The following UCSD Pascal intrinsics are for the manipulation of strings:

```
function CONCAT ( source [, source]... : string] ): string
function COPY ( source: string; index, size: integer ): string
procedure DELETE ( destination: string; index, size: integer )
procedure INSERT ( source, destination: string; size: integer )
function LENGTH ( source: string ): integer
function POS ( pattern, source: string ): integer
```

I/O Intrinsics.

READ, READLN and WRITE, WRITELN may only be used with files of type TEXT (FILE OF CHAR).

Two new file types are untyped and INTERACTIVE. In UCSD Pascal the predefined files INPUT, OUTPUT and KEYBOARD are of the type INTERACTIVE. KEYBOARD is the non-echoing equivalent of INPUT.

If a file is INTERACTIVE then

the EOF function is set by input of an <etx> character; where <etx> is defined in the file SYSTEM.MISCINFO;

the EOLN function is set by a <return>;

READ and READLN will perform a GET before loading the file's window variable; the effect of this is to require that a READ or READLN be done on an INTERACTIVE file before testing EOF or EOLN;

RESET does not load the file's window variable.

If a file is untyped then

all I/O to that file must use the BLOCKREAD and BLOCKWRITE intrinsics.

RESET and REWRITE behave as standard intrinsics, but the both may take an optional second parameter that is a disk filename -- which makes the Pascal file equivalent to the physical disk file.

Seek is an intrinsic to do random access on files. CLOSE controls the fate of a disk file. UNITREAD, UNITWRITE and other UNITxxx intrinsics are for direct control of peripheral devices. IORESULT returns the status of an I/O operation.

WRITE and WRITELN are incapable of writing Booleans or record variables. STRINGs and PACKED ARRAYs OF CHAR may be output in a single WRITE.

There are several UCSD intrinsics to handle devices and files.

```
function BLOCKREAD ( fileid: {untyped} file;
                     buffer: packed array of char;
                     blocks [, relblock]: integer ): integer

function BLOCKWRITE ( fileid: {untyped} file;
                      buffer: packed array of char;
                      blocks [, relblock]: integer ): integer

procedure CLOSE ( fileid: {any sort of} file; <option> )
      <option> ::= , LOCK | , NORMAL | , PURGE | , CRUNCH

function IORESULT : integer

procedure SEEK ( fileid: {any sort of} file; recnum: integer )

function UNITBUSY ( unitnumber: integer ): Boolean

procedure UNITCLEAR ( unitnumber: integer )

procedure UNITREAD ( unitnumber: integer;
                     buffer: packed array of char;
                     length
                     [, [blocknumber] [, option]]: integer )

procedure UNITWAIT ( unitnumber: integer )

Procedure UNITWRITE ( unitnumber: integer;
```

```
                    buffer: packed array of char;
                    length
                    [, [blocknumber] [, option]]: integer )
```

MEMORY MANAGEMENT

The memory management intrinsics only have meaning on a system with limited memory as in a micro-computer. But would have limited or no meaning on a large system with greater than 64k of memory. This will not be looked at.

CONCURRENCY

A PROCESS is declared as a procedure, and may be STARTed any number of times by the main program. Processes may be controlled by a UCSD predeclared type SEMAPHORE which is in the subrange [0..maxint].

```
procedure ATTACH ( sem: semaphore; vector: integer )

procedure SEMINIT ( var sem: semaphore; sem_count: integer )

procedure SIGNAL ( var sem: semaphore )

procedure START ( <process call>;
                  [, id: processid;]
                  [, stacksize: integer;]
                  [, priority: byterange] )
  <process call> ::= {a normal procedure call}
  type byterange: 0..255

procedure WAIT ( var sem: semaphore )
```

MISCELLANEOUS

Syntax variations:

CASE statements fall through if no label matches the selector.

Comments may be enclosed by '{ }' or '(* *)'
the two diffenent types may be nested (one level deep).

'=' and '<>' may be used for extended array or record comparisons.

GOTOs are restricted to labels within the same block.

procedure EXIT (procid: <procedure identifier>)
may be used to immediately abort a procedure.

A length attribute defines a LONG INTEGER,
the length defines the minimum number of
digits in the integer.

procedure STR (value: integer[n]; destination: string)
is used to convert an integer into a string; usually for
the output of long integers.

PACK and UNPACK are not implemented. Packing and unpacking
is done automatically. A PACKED ARRAY OF CHAR may be
assigned, input and output as a single entity.

Packed variables may not be used as call-by-reference (var)
parameters.

Sets of subranges of integers must include only positive
integers.

Set comparisons must be be between sets of the same
underlying type.

The arctangent function may be called either ATAN or ARCTAN.

APPENDIX B

```
#include <math.h>
#include <stdio.h>
/*$T "COPYRIGHT - INMOS LIMITED - 1981 ALL RIGHTS RESERVED"*/
/*$L remout:*/
/* $R-*/
/*   VERSION 4 INTERPRETER :: C DEFINITION OF THE P_STSTEM          */
/*   AUTHOR CARLOS LYNN QUALLS                                      */
/*   KANSAS STATE UNIVERSITY                                        */
/*   SPRING SEMESTER 1983                                           */
/*   IN PARTIAL FULFILLMENT OF THE REQUIRMENTS OF A MASTER'S        */
/*   DEGREE IN COMPUTER SCIENCE                                     */
/*   This is the converted source code for the UCSD P_SYSTEM        */
/*   to allow the P_SYSTEM OPERATING SYSTEM to run under the UNIX   */
/*   OPERATING SYSTEM.                                              */


/*   VERSION 4 INTERPRETER :: PASCAL DEFINITION OF THE P_SYSTEM     */
/*   COPYRIGHT - INMOS LIMITED - 1981                               */
/*   ALL RIGHTS RESERVED                                            */
/*   AUTHOR MICHAEL HARRISON                                        */
/*   This program is intended to be a definition of the version 4   */
/*   p_machine.  In any case where there is a choice between execut-*/
/*   ion efficiency, or code compactness and clarity of expression  */
/*   then clarity takes priority.  It is intended to be an unambig- */
/*   uous definition of functionality rather than a guide to        */
/*   efficient implementation                                       */

#define TRUE 1
#define FALSE 0

/*CONST*/
/*store addresses*/
#define MWA 16001           /*maximum word ADDRESS, including NIL_ptr*/
#define MBA 32000           /*maximum byte ADDRESS*/
#define mem_base 0          /*store base*/
#define mem_top 16000       /*top valid word ADDRESS*/
#define NIL_ptr 16001       /*undefined pointer*/

/*input/output constants*/
#define sys_no 4            /*system disc unit number*/
#define max_unit 16         /*maximum available unit number*/
#define fblk_size 512       /*size of disc blocking in bytes*/

/*miscellaneous constants*/
#define UNDEF -32768        /*undefined value*/
#define h_order 0           /*high order byte index*/
#define l_order 1           /*low order byte index*/
#define disp0 4             /*size of mark stack control word*/

/*file dictionary*/
#define dir_index 26        /*size of directory entry in bytes*/
#define max_dir 77          /*maximum directory entry*/
#define dfirst_block 0      /*offset contains first block number of entry*/
#define dlast_block 77      /*offset contains last block number of entry*/
```

```
#define dtid 6              /*byte offset of file identity*/
#define tid_leng 15         /*byte length of file identity*/

/* ************************************************************** */
/*segment dictionary see 3.2.A of the Internal Architecture Guide*/
/* ************************************************************** */
#define disc_info 0         /*offset to disc information per segment*/
#define disc_index 2        /*size of disc info. entry*/
#define cod_adr 0           /*segment starting block within disc info.*/
#define cod_len 1           /*number of words in segment*/
#define xseg_name 64        /*segment name offset in bytes*/
#define name_index 8        /*size of name in bytes*/
#define max_dic_seg 15      /*maximum segment number*/
#define seg_famly 144       /*offset to segment family entry*/

/* ************************************************************** */
/*Segment Information Blocks - see section 3.4 of Architecture Guide*/
/* ************************************************************** */
#define  act_val 6      /*activity constant used to update mem activity count*/
#define sib_base 140    /*position of base SIB in store*/
#define sib_index 17    /*size of SIB*/
#define sib_lim 25      /*maximum SIB number*/

/* ************************************************************** */
/*sib constants, to map onto an array                             */
/* ************************************************************** */
#define seg_base 0          /*offset contains base memory location*/
#define s_ref_count 1       /*number of active calls*/
#define s_activity 2        /*memory swap activity*/
#define s_link_count 3      /*number of links to SIB*/
#define s_residency 4       /*-1=pos lock, 0=swap, n=memlock*/
#define seg_name 5          /*segment name [0..7] of char*/
#define seg_leng 9          /*number of words in segment*/
#define seg_addr 10         /*disc ADDRESS of segment*/
#define seg_unit 11         /*disc unit of segment*/
#define DATA_SIZE 12        /*no. of words in data segment*/
#define next_sib 13         /*next SIB in list*/
#define prev_sib 14         /*previous SIB in list*/
#define sort_sib 15         /*next SIB in sort list*/
#define new_loc 15          /*temp ADDRESS*/
#define mtype 16            /*interpreter dependent*/

/* ************************************************************** */
/*Event Records see section 3.5 of Architecture Guide             */
/*erec constants to make event record map on memory space         */
/* ************************************************************** */
#define GBLVEC 625          /*location of pointer to EVEC*/
#define EV_index 5          /*size of EREC*/
#define EREC_lim 25         /*maximum EREC*/
#define EREC_base 14        /*location of first EREC*/
#define env_data 0          /*offset in record points to global data*/
#define e_vect 1            /*pointer to EVEC*/
#define env_sib 2           /*pointer to SIB for segment number*/
#define e_link_count 3      /*number of links to EREC*/
```

```
#define e_next_rec 4          /*next environment record*/

/* ********************************************************************** */
/*code segment offsets                                                    */
/* ********************************************************************** */
#define re_list 1             /*pointer to relocation list*/
#define seg_bsiw 6            /*offset to byte sex indicator*/
#define seg_coff 7            /*pointer to constant pool*/

/*stack factor*/
#define proc_slp 40           /*procedure slack factor*/

/* ********************************************************************** */
/*sys-com area mapping onto memory space                                  */
/* ********************************************************************** */
#define sys_base 565          /*position of sys_com in store*/
#define syscom_index 53       /*size of sys_com*/

/*offsets within sys_com*/
#define io_rslt 0             /*result of last io call*/
#define xeq_error 1           /*reason for exec error call*/
#define sys_unit 2            /*unit number for system device*/
#define rw_table 3            /*pascal reserved words for id search*/
#define gdir_p 4              /*global directory pointer*/
#define flt_sem 7             /*fault semaphores*/
#define flt_tib 9             /*pointer to TIB at fault time*/
#define flt_rec 10            /*pointer to EREC to leave in memory*/
#define flt_wds 11            /*number of words needed*/
#define flt_num 12            /*fault number*/
#define lo_time 30            /*time*/
#define hi_time 31
#define misc_info 33          /*crt information*/
#define crt_type 34
#define crt_ctl 35
#define crt_nfo 41            /*height of crt*/
#define width 42              /*width of crt*/
#define syeof 45              /*control character information*/

/* ********************************************************************** */
/*semaphore information                                                   */
/* ********************************************************************** */
#define sm_count 0            /*semaphore count*/
#define sm_queue 1            /*tasks queued on semaphore*/

/* ********************************************************************** */
/*task information block                                                  */
/* ********************************************************************** */
#define tail 0                /*pointer to next tib*/
#define tpri 2                /*priority of task, byte offset*/
#define tlo_stack 2           /*task low stack limit*/
#define thi_stack 3           /*task high stack limit-ie bottom of stack*/
#define tsp 4                 /*task stack pointer*/
#define tmpd0 5               /*task L register*/
#define filler1 6
```

```
#define tpc 7                   /*task program counter*/
#define terec 8                 /*task current EREC*/
#define proc_n 18               /*task current procedure number byte offset*/
#define tio_r 19                /*task io result*/
#define thang 10                /**/
#define filler3 11

/* ********************************************************************** */
/*error codes which communicate out                                       */
/*Note that these values are twice the standard value - for               */
/*implementation reasons                                                  */
/* ********************************************************************** */
#define OK 1                    /*no error*/
#define inv_ndx 2               /*invalid index*/
#define no_proc 4               /*non existent segment*/
#define no_exit 6               /*exiting procedure never called*/
#define int_over 10             /*integer overflow*/
#define div_zer 12              /*divide by zero*/
#define bad_mem 14              /*bad memory access*/
#define u_break 16              /*user break*/
#define sy_ioer 18              /*system io error*/
#define io_error 20             /*user io error*/
#define UNIMP 22                /*instruction not implemented*/
#define fpierr 24               /*floating point error*/
#define string_fault 26         /*string too long*/
#define hlt 28                  /*unconditional halt*/
#define bpt_hlt 30              /*break point halt*/

/*error codes above 130 are implementation defined*/
#define seg_fault 256           /*segment fault*/
#define stack_fault 258         /*stack overflow*/
#define task_sw 260             /*task swap*/
#define ret_bpt 262             /*return from break point*/
#define level_error 264         /*lex level error in proc call*/
#define set_fault 266           /*set operation error*/
#define native_mode 268         /*native mode entry*/
#define run_error 270           /*other run time error*/
/* ********************************************************************** */

/*
TYPE
*/
typedef short small_type;
typedef short code_type;
typedef short ADDRESS;          /* word ADDRESS range */
typedef short BYTE_ADDRESS;     /* byte ADDRESS range */
typedef short BOOLEAN;

#define funct_code 259              /*operation code type*/
#define SLDC0      0
#define SLDC1      1
#define SLDC2      2
#define SLDC3      3
#define SLDC4      4
```

```
#define SLDC5      5
#define SLDC6      6
#define SLDC7      7
#define SLDC8      8
#define SLDC9      9
#define SLDC10    10
#define SLDC11    11
#define SLDC12    12
#define SLDC13    13
#define SLDC14    14
#define SLDC15    15
#define SLDC16    16
#define SLDC17    17
#define SLDC18    18
#define SLDC19    19
#define SLDC20    20
#define SLDC21    21
#define SLDC22    22
#define SLDC23    23
#define SLDC24    24
#define SLDC25    25
#define SLDC26    26
#define SLDC27    27
#define SLDC28    28
#define SLDC29    29
#define SLDC30    30
#define SLDC31    31

#define SLDL1     32
#define SLDL2     33
#define SLDL3     34
#define SLDL4     35
#define SLDL5     36
#define SLDL6     37
#define SLDL7     38
#define SLDL8     39
#define SLDL9     40
#define SLDL10    41
#define SLDL11    42
#define SLDL12    43
#define SLDL13    44
#define SLDL14    45
#define SLDL15    46
#define SLDL16    47

#define SLDO1     48
#define SLDO2     49
#define SLDO3     50
#define SLDO4     51
#define SLDO5     52
#define SLDO6     53
#define SLDO7     54
#define SLDO8     55
#define SLDO9     56
```

```
#define SLDO10    57
#define SLDO11    58
#define SLDO12    59
#define SLDO13    60
#define SLDO14    61
#define SLDO15    62
#define SLDO16    63

#define UNIM1     64
#define UNIM2     65
#define UNIM3     66
#define UNIM4     67
#define UNIM5     68
#define UNIM6     69
#define UNIM7     70
#define UNIM8     71
#define UNIM9     72
#define UNIM10    73
#define UNIM11    74
#define UNIM12    75
#define UNIM13    76
#define UNIM14    77
#define UNIM15    78
#define UNIM16    79
#define UNIM17    80
#define UNIM18    81
#define UNIM19    82
#define UNIM20    83
#define UNIM21    84
#define UNIM22    85
#define UNIM23    86
#define UNIM24    87
#define UNIM25    88
#define UNIM26    89
#define UNIM27    90
#define UNIM28    91
#define UNIM29    92
#define UNIM30    93
#define UNIM31    94
#define UNIM32    95

#define SLLA1     96
#define SLLA2     97
#define SLLA3     98
#define SLLA4     99
#define SLLA5    100
#define SLLA6    101
#define SLLA7    102
#define SLLA8    103

#define SSTL1    104
#define SSTL2    105
#define SSTL3    106
#define SSTL4    107
```

```
#define SSTL5     108
#define SSTL6     109
#define SSTL7     110
#define SSTL8     111

#define SCGX1     112
#define SCGX2     113
#define SCGX3     114
#define SCGX4     115
#define SCGX5     116
#define SCGX6     117
#define SCGX7     118
#define SCGX8     119

#define SIND0     120
#define SIND1     121
#define SIND2     122
#define SIND3     123
#define SIND4     124
#define SIND5     125
#define SIND6     126
#define SIND7     127

#define LDCB      128
#define LDCI      129
#define LCO       130
#define LDC       131
#define LLA       132
#define LDO       133
#define LAO       134
#define LDL       135
#define LDA       136
#define LOD       137
#define UJP       138
#define UJPL      139
#define MPI       140
#define DVI       141
#define STM       142
#define MODI      143
#define CLP       144
#define CGP       145
#define CIP       146
#define CLX       147
#define CGX       148
#define CIX       149
#define RPU       150
#define CFP       151
#define LDCN      152
#define LSL       153
#define LDE       154
#define LAE       155
#define NOP       156
#define LPR       157
#define BPT       158
```

```
#define BNOT      159
#define LOR       160
#define LAND      161
#define ADI       162
#define SBI       163
#define STL       164
#define SRO       165
#define STR       166
#define LDB       167
#define NAT       168
#define NAT_INFO  169
#define UNIM59    170
#define CAP       171
#define CSP       172
#define SLOD1     173
#define SLOD2     174
#define UNIM62    175
#define EQUI      180
#define NEQI      181
#define LEQI      182
#define GEQI      183
#define LEQU      184
#define GEQU      185
#define EQPWR     186
#define LEPWR     187
#define GEPWR     188
#define EQBYT     189
#define LEBYT     190
#define GEBYT     191
#define SRS       192
#define SWAP      193
#define TNC       194
#define RND       195
#define ADR       196
#define SBR       197
#define MPR       198
#define DVR       199
#define STO       200
#define MOV       201
#define DUP2      202
#define ADJ       203
#define STB       204
#define LDP       205
#define STP       206
#define CHK       207
#define FLT       208
#define EQREAL    209
#define LEREAL    210
#define GEREAL    211
#define LDM       212
#define SPR       213
#define EFJ       214
#define NFJ       215
#define FJP       216
```

```
#define FJPL     217
#define XJP      218
#define IXA      219
#define IXP      220
#define STE      221
#define INN      222
#define UNI      223
#define INT      224
#define DIF      225
#define SIG      226
#define WAT      227
#define ABI      228
#define NGI      229
#define DUP1     230
#define ABR      231
#define NGR      232
#define LNOT     233
#define IND      234
#define INCR     235
#define EQSTR    236
#define LESTR    237
#define GESTR    238
#define ASTR     239
#define CSTR     240
#define INCI     241
#define DECI     242
#define SCIP1    243
#define SCIP2    244
#define TJP      245
#define LDCRL    246
#define LDRL     247
#define STRL     248
#define CTRL     249
#define EXPRL    250
#define UNIM73   251
#define UNIM74   252
#define UNIM75   253
#define UNIM76   254
#define UNIM77   255
#define UNIM78   256
#define UNIM79   257
#define UNIM80   258
#define PAUSE    259

/*   par_type =              parameter type*/
#define NONE        0
#define UBP         1
#define WP          2
#define BP          3
#define DB_B        4
#define UB_B        5
#define UB1_B_UB2   6
#define UB1_UB2     7
#define UB1_UB2_B   8
```

```
#define SBP          9
#define DB_UB       10
#define UB1_DB_UB2  11
#define DBP         12
#define NOT_IMP     13

/*various puns follow in order to use store in different ways*/
union pun_byte
{
        short byte_intf;
        char  bytf[2];
};

union pun_set
{
        unsigned short setf;
        unsigned short set_intf;
};

union pun_real
{
        float realf;
        short real_intf[2];
};

union pun_bool
{
        short boolf;
        short bool_intf;
};

union pun_char
{
        char charf;
        short char_intf;
};

union pun_address
{
        ADDRESS addf;
        short address_intf;
};

union pun_byddr
{
        BYTE_ADDRESS badf;
        short byddr_intf;
};

struct proc_rec
{
        short data_size;
        BYTE_ADDRESS code_start;
        BYTE_ADDRESS code_exit;
```

```
};

/*
VAR
*/
/* P_machine registers in capitals */
short
G, L,                                   /*global and local base registers*/
SP,PC,                                  /*stack pointer and program counter*/
old_PC,                                 /*previous program counter*/
UB, UB1, UB2,                           /*unsigned byte parameter locations*/
SB,                                     /*signed byte parameter location*/
DB,                                     /*byte parameter used when intermediates*/
B,W,                                    /*word parameters*/
error,                                  /*global error variable*/
operator,                               /*operation code for current instruction*/
READY_Q, CTASK,                         /*pointer to task ready queue, and
                                            pointer to current task block*/
EVEC,                                   /*pointer to EREC vector*/
EREC,                                   /*pointer to current event record*/
old_EREC,flt_EREC,                      /*pointers to old EREC, and EREC for
                                            restoration after fault*/
seg_bot, old_segbot,                    /*pointer to current and previous
                                            code segment bases*/
sibp,                                   /*pointer to current segment
                                            information block*/
last_frame,                             /*pointer to last stack frame*/
procnumber;                             /*current procedure number*/
struct proc_rec proc_details;           /*procedure details for use in
                                            procedure call*/
short res_tbl[25];                      /*access table to identity search*/

struct                                    /*identity search table see IDS*/
{
        char word[9];
        short token[2];
}
*search_table[43];

char sys_name[14];                       /*identity of system file*/
short par_table[funct_code];             /*parameter configs for ops*/

/* ********************************************************************** */
/* the following data structure defines p_machine store - store has a  */
/* variety of types imposed on it.  For this reason the data structure */
/* is punned to permit access of 16 bit integers-INTEGER; bytes, 8 bit */
/* characters-fchar, operator codes-func and 16 bit sets-fset          */
/* ********************************************************************** */
union store_structure
{
        short           data  [MWA];
        char            code  [MBA];
        char            fchar [MBA];
        char            func  [MBA];
```

```
        unsigned short   fset  [MWA];
}
store;

union swapper
{
        short sint;
        char  schar[2];
}
swap1,swap2,swap3;
```

```
main() /*INTERPRETER*/
{
        error = OK;
        init_const();
        house_keep();
        boot_strap();
        set_ptable();
        printf("Ready to go0);
        while (error==OK)
        {
                fetch();
                decode();
                if (error!=OK)
                {
                        while ((error == task_sw)   ||
                            (error == seg_fault) ||
                            (error == stack_fault))
                        {
                            switch (error)
                            {
                            case stack_fault:
                                    {
                                            error = OK;
                                            PC = old_PC;
                                            /*end loop*/
                                            signal(sys_base+flt_sem);
                                            break;
                                    };

                            case seg_fault:
                                    {
                                            /*see CGX*/
                                            error = OK;
                                            PC = old_PC;
                                            push(flt_EREC);
                                            push(NIL_ptr);
                                            push(seg_fault / 2);
                                            seg_and_proc(1,2,G-disp0);
                                            if (error==stack_fault)
                                                    contract(3);
                                            else
                                            if (error!=OK)
                                                    error = run_error;
                                            break;
                                    };

                            case task_sw:
                                    {
                                            error = OK;
                                            save_context();
                                            store.data[sibp+s_activity] =
                                            store.data[sibp+s_activity]
                                            + act_val;
                                            READY_Q = queue(READY_Q,CTASK);
```

```
                                                run_task();
                                                break;
                                        };
                                };/*switch*/
                        };/*while*/
                        if (error!=OK)
                        {
                                push(NIL_ptr);
                                push(NIL_ptr);
                                push(error / 2);
                                error = OK;
                                seg_and_proc(1,2,G-disp0);
                        };/*IF*/
                };/*IF*/
        };/*WHILE*/
        printf("illegal operation = %4d0, operator);
}
```

```
init_const()
/*this procedure initializes various tables*/
{
        strcpy(sys_name , "SYSTEM.PASCAL");

        strcpy(search_table[0]->word , "AND     ");
        search_table[0]->token[0] = 39;
        search_table[0]->token[1] = 2;

        strcpy(search_table[1]->word , "ARRAY   ");
        search_table[1]->token[0] = 44;
        search_table[1]->token[1] = 15;

        strcpy(search_table[2]->word , "BEGIN   ");
        search_table[2]->token[0] = 19;
        search_table[2]->token[1] = 15;

        strcpy(search_table[3]->word , "CASE    ");
        search_table[3]->token[0] = 21;
        search_table[3]->token[1] = 15;

        strcpy(search_table[4]->word , "CONST   ");
        search_table[4]->token[0] = 28;
        search_table[4]->token[1] = 15;

        strcpy(search_table[5]->word , "DIV     ");
        search_table[5]->token[0] = 39;
        search_table[5]->token[1] = 3;

        strcpy(search_table[6]->word , "DO      ");
        search_table[6]->token[0] = 6;
        search_table[6]->token[1] = 15;

        strcpy(search_table[7]->word , "DOWNTO  ");
        search_table[7]->token[0] = 8;
        search_table[7]->token[1] = 15;

        strcpy(search_table[8]->word , "ELSE    ");
        search_table[8]->token[0] = 13;
        search_table[8]->token[1] = 15;

        strcpy(search_table[9]->word , "END     ");
        search_table[9]->token[0] = 9;
        search_table[9]->token[1] = 15;

        strcpy(search_table[10]->word , "EXTERNAL");
        search_table[10]->token[0] = 53;
        search_table[10]->token[1] = 15;

        strcpy(search_table[11]->word , "FOR     ");
        search_table[11]->token[0] = 24;
        search_table[11]->token[1] = 15;

        strcpy(search_table[12]->word , "FILE    ");
```

```
search_table[12]->token[0] = 46;
search_table[12]->token[1] = 15;

strcpy(search_table[13]->word , "FORWARD ");
search_table[13]->token[0] = 34;
search_table[13]->token[1] = 15;

strcpy(search_table[14]->word , "FUNCTION");
search_table[14]->token[0] = 32;
search_table[14]->token[1] = 15;

strcpy(search_table[15]->word , "GOTO    ");
search_table[15]->token[0] = 26;
search_table[15]->token[1] = 15;

strcpy(search_table[16]->word , "IF      ");
search_table[16]->token[0] = 20;
search_table[16]->token[1] = 15;

strcpy(search_table[17]->word , "IMPLEMEN");
search_table[17]->token[0] = 52;
search_table[17]->token[1] = 15;

strcpy(search_table[18]->word , "IN      ");
search_table[18]->token[0] = 41;
search_table[18]->token[1] = 14;

strcpy(search_table[19]->word , "INTERFAC");
search_table[19]->token[0] = 51;
search_table[19]->token[1] = 15;

strcpy(search_table[20]->word , "LABEL   ");
search_table[20]->token[0] = 27;
search_table[20]->token[1] = 15;

strcpy(search_table[21]->word , "MOD     ");
search_table[21]->token[0] = 39;
search_table[21]->token[1] = 4;

strcpy(search_table[22]->word , "NOT     ");
search_table[22]->token[0] = 38;
search_table[22]->token[1] = 15;

strcpy(search_table[23]->word , "OF      ");
search_table[23]->token[0] = 11;
search_table[23]->token[1] = 15;

strcpy(search_table[24]->word , "OR      ");
search_table[24]->token[0] = 40;
search_table[24]->token[1] = 7;

strcpy(search_table[25]->word , "PACKED  ");
search_table[25]->token[0] = 43;
search_table[25]->token[1] = 15;
```

```
strcpy(search_table[26]->word , "PROCEDUR");
search_table[26]->token[0] = 31;
search_table[26]->token[1] = 15;

strcpy(search_table[27]->word , "PROCESS ");
search_table[27]->token[0] = 56;
search_table[27]->token[1] = 15;

strcpy(search_table[28]->word , "PROGRAM ");
search_table[28]->token[0] = 33;
search_table[28]->token[1] = 15;

strcpy(search_table[29]->word , "RECORD  ");
search_table[29]->token[0] = 45;
search_table[29]->token[1] = 15;

strcpy(search_table[30]->word , "REPEAT  ");
search_table[30]->token[0] = 22;
search_table[30]->token[1] = 15;

strcpy(search_table[31]->word , "SET     ");
search_table[31]->token[0] = 42;
search_table[31]->token[1] = 15;

strcpy(search_table[32]->word , "SEGMENT ");
search_table[32]->token[0] = 33;
search_table[32]->token[1] = 15;

strcpy(search_table[33]->word , "SEPARATE");
search_table[33]->token[0] = 54;
search_table[33]->token[1] = 15;

strcpy(search_table[34]->word , "THEN    ");
search_table[34]->token[0] = 12;
search_table[34]->token[1] = 15;

strcpy(search_table[35]->word , "TO      ");
search_table[35]->token[0] = 7;
search_table[35]->token[1] = 15;

strcpy(search_table[36]->word , "TYPE    ");
search_table[36]->token[0] = 29;
search_table[36]->token[1] = 15;

strcpy(search_table[37]->word , "UNIT    ");
search_table[37]->token[0] = 50;
search_table[37]->token[1] = 15;

strcpy(search_table[38]->word , "UNTIL   ");
search_table[38]->token[0] = 10;
search_table[38]->token[1] = 15;

strcpy(search_table[39]->word , "USES    ");
```

```
        search_table[39]->token[0] = 49;
        search_table[39]->token[1] = 15;

        strcpy(search_table[40]->word , "VAR     ");
        search_table[40]->token[0] = 30;
        search_table[40]->token[1] = 15;

        strcpy(search_table[41]->word , "WHILE   ");
        search_table[41]->token[0] = 23;
        search_table[41]->token[1] = 15;

        strcpy(search_table[42]->word , "WITH    ");
        search_table[42]->token[0] = 25;
        search_table[42]->token[1] = 15;
```

```
        /*set up res_tbl*/
        res_tbl[0]  = 0;
        res_tbl[1]  = 2;
        res_tbl[2]  = 3;
        res_tbl[3]  = 5;
        res_tbl[4]  = 8;
        res_tbl[5]  = 11;
        res_tbl[6]  = 15;

        res_tbl[7]  = 16;
        res_tbl[8]  = 16;

        res_tbl[9]  = 20;
        res_tbl[10] = 20;
        res_tbl[11] = 20;

        res_tbl[12] = 21;
        res_tbl[13] = 22;
        res_tbl[14] = 23;
        res_tbl[15] = 25;

        res_tbl[16] = 29;
        res_tbl[17] = 29;

        res_tbl[18] = 31;
        res_tbl[19] = 34;
        res_tbl[20] = 37;
        res_tbl[21] = 40;
        res_tbl[22] = 41;
        res_tbl[23] = 43;

        res_tbl[24] = 44;
        res_tbl[25] = 44;
};
```

```
house_keep()
{
        short i,j;
        char temp_ptr;

        PC = mem_base;
        SP = mem_top;

        /*set up a special root TIB*/
        store.data[mem_base+tail] = NIL_ptr;
        store.code[mem_base*2+tpri] = 128;          /*root task priority*/
        store.data[mem_base+tlo_stack] = 0;         /*lower stack limit*/
        store.data[mem_base+thi_stack] = mem_top;   /*upper stack limit*/
        store.data[mem_base+tsp] = NIL_ptr;         /*stack pointer*/
        store.data[mem_base+tmpd0] = NIL_ptr;       /*local base*/
        store.data[mem_base+filler1] = NIL_ptr;
        store.data[mem_base+tpc] = -1;              /*program counter*/
        store.data[mem_base+terec] = NIL_ptr;       /*event record*/
        store.code[mem_base*2+proc_n] = 0;          /*procedure number*/
        store.code[mem_base*2+tio_r] = 0;           /*i/o error*/
        store.data[mem_base+thang] = NIL_ptr;
        store.data[mem_base+filler3] = NIL_ptr;
        store.data[mem_base+12] = 1;                /*mail task*/
        store.data[mem_base+13] = NIL_ptr;          /*MSCW*/

        /*set up initial EVEC*/
        EVEC = GBLVEC;
        store.data[GBLVEC] = 25;

        /*set up global bases*/
        READY_Q = NIL_ptr;
        CTASK = mem_base;                       /*points to the root TIB*/
        sibp = sib_base + sib_index*(16-1);     /*points to sib 16*/
        EREC = EREC_base + EV_index*(16-1);     /*points to EREC for seg 16*/

        /*set up ERECs*/
        temp_ptr = EREC_base;
        for (i = 0; i <= 24; i++)
        {
                store.data[GBLVEC+i+1] = temp_ptr;      /*initialize EVEC*/
                /*points to first location in heap*/
                store.data[temp_ptr+env_data] = GBLVEC+25;
                store.data[temp_ptr+e_vect] = EVEC;
                store.data[temp_ptr+env_sib] = sib_base + sib_index*i;
                store.data[temp_ptr+e_link_count] = 0;
                store.data[temp_ptr+e_next_rec] = NIL_ptr;
                temp_ptr = temp_ptr + EV_index;
        };
        /*G is first location in heap plus disp0*/
        G = store.data[EREC+env_data]+disp0;
        /*pc set up to bootstrap area*/
        PC = G*2;
        /*set up SIB's*/
        temp_ptr = sib_base;
```

```
        for (i = 0; i <= 24; i++)
        {
                store.data[temp_ptr+seg_base] = NIL_ptr;
                store.data[temp_ptr+s_ref_count] = 0;    /*refs*/
                store.data[temp_ptr+s_activity] = 0;
                store.data[temp_ptr+s_link_count] = 1;
                store.data[temp_ptr+s_residency] = 0;
                for (j = 0; j <= 3; j++)
                        store.data[temp_ptr+seg_name+j] = 0;
                store.data[temp_ptr+seg_leng] = 0;
                store.data[temp_ptr+seg_addr] = 0;
                store.data[temp_ptr+seg_unit] = sys_no;
                store.data[temp_ptr+DATA_SIZE] = 0;
                store.data[temp_ptr+next_sib] = NIL_ptr;
                store.data[temp_ptr+prev_sib] = NIL_ptr;
                store.data[temp_ptr+sort_sib] = 0;

                temp_ptr = temp_ptr+sib_index;
        };

        store.data[sib_base+s_ref_count] = 1;
        store.data[sib_base+s_residency] = -1;
```

```
        /* ************************************************************ */
        /*set up SYS COM                                                */
        /*crt control values are set up to zero by the following loop-*/
        /*controls which have values are explicitly loaded see below  */
        /* ************************************************************ */
        for (i = 0; i <= 33; i++)
                store.data[sys_base+i] = 0;
        store.data[sys_base+sys_unit] = 4;
        store.data[sys_base+gdir_p] = NIL_ptr;
        store.data[sys_base+crt_type] = 3;
        temp_ptr = (sys_base+crt_nfo)*2;
        store.code[temp_ptr+1] = 13;                /*home*/
        store.code[temp_ptr+6] = 10;                /*fill count*/
        store.data[sys_base+width] = 72;            /*width of crt*/

        temp_ptr = (sys_base+syeof)*2;
        store.code[temp_ptr] = 3;                   /*end of file*/
        store.code[temp_ptr+1] = 6;                 /*flush line*/
        store.code[temp_ptr+2] = 0;                 /*break*/
        store.code[temp_ptr+3] = 19;                /*stop output*/
        store.code[temp_ptr+4] = 95;                /*delete*/
        store.code[temp_ptr+5] = 63;                /*bad character*/
        store.code[temp_ptr+6] = 127;               /*line delete*/
        store.code[temp_ptr+7] = 27;                /*escape*/
        store.code[temp_ptr+8] = 0;                 /*prefix*/
        store.code[temp_ptr+9] = 3;                 /*etx*/
        store.code[temp_ptr+10] = 8;                /*backspace*/
        store.code[temp_ptr+11] = 18;               /*alphalock*/
};
```

```
boot_strap()
{
        int fd;
        short n,j,i,bytes_read;
        ADDRESS dir_limit,temp_ptr;
        BOOLEAN t,OS_byte_swapped;
        ADDRESS sys_bno;
        /*get the time?*/
        /*step 1 in the bookstrap*/
        /*get the directory*/
        fd = open("adapz.d1",0);
        /*************************************************************/
        /* the number 3328 in the following lseek statement was      */
        /* calculated from FIGURE 4 page 20 of the UCSD p-System     */
        /* Installation Guide titled "Layout of an Adaptable         */
        /* System Logical Disk As Distributed by SofTech             */
        /* Microsystems"                                             */
        /*************************************************************/
        lseek(fd,(long)(3328+(2*512)),0);
        bytes_read = read(fd,&store.code[PC],4*512);
        /*treat the block read in as a directory*/

        dir_limit = PC+dir_index*max_dir;
        temp_ptr = PC+1;

        /*step 2 locate the operating system code file*/
        do
            {
                t = TRUE;
                temp_ptr = temp_ptr+dir_index;
                t = strncmp(sys_name,&store.fchar[temp_ptr+dtid],13);
        }
        while ((t!=0) && (temp_ptr<=dir_limit));

        if(t==0)
                /*found SYSTEM.PASCAL*/
        {
                /*******************************************************/
                /* Need to be able to check here to see if the file    */
                /* directory on                                        */
                /* the disk you are booting from is byte swapped       */
                /* or not.                                             */
                /*******************************************************/
                if(store.data[(PC/2)+1] > 256)
                {
                        /* The directory is byte swapped      */
                        /* must swap bytes before calulations */
                        swap1.sint = contents((temp_ptr+dfirst_block) / 2);
                        swap2.schar[0] = swap1.schar[1];
                        swap2.schar[1] = swap1.schar[0];
                        sys_bno = swap2.sint;
                }
                else sys_bno = contents((temp_ptr+dfirst_block) / 2);
                lseek(fd,(long)(3328+(sys_bno*512)),0);
```

```
bytes_read = read(fd,&store.code[PC],1*512);
/*step 3 initialize data structures for segments */
/*which are in the first segment dictionary block*/
/*of the OS code file                             */

/*treat the block as a segment dictionary*/
/*************************************************/
/* Need to check here to see if the first segment */
/* dictionary block of the OS code file is byte   */
/* swapped or not.                                */
/*************************************************/
if(store.data[(PC/2)+255]==256)
        OS_byte_swapped = 1;
else OS_byte_swapped = 0;
temp_ptr = sib_base;
/*************************************************/
/* Creating a segment dictionary as per pages 27-32 */
/* in the INTERNAL ARCHITECTURAL GUIDE              */
/*************************************************/
for (i = 0; i <= max_dic_seg; i++)
{
        for (j = 0; j <= 7; j++)
                store.fchar[(temp_ptr+seg_name)*2+j] =
                store.fchar[PC + xseg_name+(i*name_index)+j];
        if(OS_byte_swapped ==1)
        {
                swap1.sint =
                store.data[PC/2+disc_info+(i*disc_index)
                + cod_len];
                swap2.schar[0] = swap1.schar[1];
                swap2.schar[1] = swap1.schar[0];
                store.data[temp_ptr+seg_leng] = swap2.sint;
                swap1.sint =
                store.data[PC/2+disc_info+(i*disc_index)
                + cod_adr];
                swap2.schar[0] = swap1.schar[1];
                swap2.schar[1] = swap1.schar[0];
                store.data[temp_ptr+seg_addr] =
                swap2.sint+sys_bno;
        }
        else
        {
                store.data[temp_ptr+seg_leng] =
                store.data[PC / 2+disc_info+(i*disc_index)
                + cod_len];
                store.data[temp_ptr+seg_addr] =
                store.data[PC / 2+disc_info+(i*disc_index)
                + cod_adr] + sys_bno;
        }

        temp_ptr = temp_ptr+sib_index;
};

/*step 4 load kernel ie seg 1 into top of memory*/
```

```
                n = store.data[sib_base+seg_leng];
                SP = SP-n-1;
                lseek(fd,(long)(3328+(store.data[sib_base+seg_addr]*512)),0);
                bytes_read = read(fd,&store.code[SP*2],n*2);
                store.data[sib_base+seg_base] = SP;
                /*set up activation record*/
                push(1); /*dummy procedure number*/
                push(NIL_ptr);
                push(NIL_ptr);
                push(SP-4);
                push(SP-2);
                last_frame = SP;
                L = last_frame+4;
                /*push segment 16 ie boot segment onto stack*/
                temp_ptr = sib_base + sib_index*(16-1);
                n = store.data[temp_ptr+seg_leng];
                SP = SP-n-1;
                seg_bot = SP;
                lseek(fd,(long)3328+(store.data[temp_ptr+seg_addr]*512),0);
                bytes_read = read(fd,&store.code[seg_bot*2],n*2);
                store.data[temp_ptr+seg_base] = seg_bot;
                old_segbot = mem_top;
                if(OS_byte_swapped == 1)
                {
                        swap1.sint = store.data[seg_bot];
                        swap2.schar[0] = swap1.schar[1];
                        swap2.schar[1] = swap1.schar[0];
                        temp_ptr = swap2.sint;
                }
                else temp_ptr = store.data[seg_bot];
                if(OS_byte_swapped == 1)
                {
                        swap1.sint = store.data[G+seg_famly];
                        swap2.schar[0] = swap1.schar[1];
                        swap2.schar[1] = swap1.schar[0];
                        store.data[mem_base+tlo_stack] =
                            G+swap2.sint+1;/*remember G=data seg base + disp0*/
                        /*so add 1 to 'correct' to size of(mscw)*/
                }
                else store.data[mem_base+tlo_stack] =
                     G + store.data[G+seg_famly] + 1;
                store.data[mem_base+thi_stack] = SP; /*set up stack limits*/
                if(OS_byte_swapped == 1)
                {
                        swap1.sint = store.data[seg_bot+temp_ptr-1];
                        swap2.schar[0] = swap1.schar[1];
                        swap2.schar[1] = swap1.schar[0];
                        PC = seg_bot+swap2.sint+1;
                }
                else PC = seg_bot+store.data[seg_bot+temp_ptr-1]+1;
                PC = PC*2;
                /*assumes proc_no 0*/
        }
        else
```

```
        {
                error = io_error;
                printf("SYSTEM.PASCAL not found0);
        }
        /*3 :*/
        store.data[G+1] = sys_base;      /*set up ptr to syscom*/
        store.data[G-disp0] = G-disp0;
};
```

```
/*set up the parameter table*/
set_ptable()
{
        for (operator = SLDC0; operator <= PAUSE; operator++)
        {
                par_table[operator] = NOT_IMP;
                switch (operator) {
                case SLDC0:
                case SLDC1:
                case SLDC2:
                case SLDC3:
                case SLDC4:
                case SLDC5:
                case SLDC6:
                case SLDC7:
                case SLDC8:
                case SLDC9:
                case SLDC10:
                case SLDC11:
                case SLDC12:
                case SLDC13:
                case SLDC14:
                case SLDC15:
                case SLDC16:
                case SLDC17:
                case SLDC18:
                case SLDC19:
                case SLDC20:
                case SLDC21:
                case SLDC22:
                case SLDC23:
                case SLDC24:
                case SLDC25:
                case SLDC26:
                case SLDC27:
                case SLDC28:
                case SLDC29:
                case SLDC30:
                case SLDC31:

                case SLDL1:
                case SLDL2:
                case SLDL3:
                case SLDL4:
                case SLDL5:
                case SLDL6:
                case SLDL7:
                case SLDL8:
                case SLDL9:
                case SLDL10:
                case SLDL11:
                case SLDL12:
                case SLDL13:
                case SLDL14:
```

```
case SLDL15:
case SLDL16:

case SLD01:
case SLD02:
case SLD03:
case SLD04:
case SLD05:
case SLD06:
case SLD07:
case SLD08:
case SLD09:
case SLD010:
case SLD011:
case SLD012:
case SLD013:
case SLD014:
case SLD015:
case SLD016:

case SLLA1:
case SLLA2:
case SLLA3:
case SLLA4:
case SLLA5:
case SLLA6:
case SLLA7:
case SLLA8:

case SSTL1:
case SSTL2:
case SSTL3:
case SSTL4:
case SSTL5:
case SSTL6:
case SSTL7:
case SSTL8:

case SIND0:
case SIND1:
case SIND2:
case SIND3:
case SIND4:
case SIND5:
case SIND6:
case SIND7:

case LDCN:
case STO:
case LDB:
case STB:
case LDP:
case STP:
case LAND:
```

```
case LOR:
case LNOT:
case LEQU:
case GEQU:
case ABI:
case NGI:
case ADI:
case SBI:
case MPI:
case DVI:
case MODI:
case CHK:
case EQUI:
case NEQI:
case LEQI:
case GEQI:
case FLT:
case TNC:
case RND:
case ABR:
case NGR:
case ADR:
case SBR:
case MPR:
case DVR:
case EQREAL:
case LEREAL:
case GEREAL:
case SRS:
case INN:
case UNI:
case INT:
case DIF:
case EQPWR:
case LEPWR:
case GEPWR:
case CFP:
case BPT:
case SIG:
case WAT:
case CSTR:
case LPR:
case SPR:
case DUP1:
case DUP2:
case SWAP:
case NOP:
case NAT:
case LDRL:
case STRL:
case EXPRL:
case CTRL:
case INCI:
case DECI:
```

```
case BNOT:

        par_table[operator] = NONE;
        break;

case LDCB:
case LDM:
case STM:
case CSP:
case ADJ:
case CLP:
case CGP:
case SCIP1:
case SCIP2:
case SCGX1:
case SCGX2:
case SCGX3:
case SCGX4:
case SCGX5:
case SCGX6:
case SCGX7:
case SCGX8:

        par_table[operator] = UBP;
        break;

case LDCI:
case UJPL:
case FJPL:

        par_table[operator] = WP;
        break;

case LCO:
case LDL:
case LLA:
case STL:
case LDO:
case LAO:
case SRO:
case IND:
case CAP:
case INCR:
case IXA:
case XJP:
case RPU:
case SLOD1:
case SLOD2:
case LDCRL:

        par_table[operator] = BP;
        break;
```

```
case LOD:
case LDA:
case STR:

        par_table[operator] = DB_B;
        break;

case LDE:
case LAE:
case STE:
case MOV:

        par_table[operator] = UB_B;
        break;

case LDC:

        par_table[operator] = UB1_B_UB2;
        break;

case IXP:
case CLX:
case CGX:
case EQSTR:
case LESTR:
case GESTR:
case ASTR:

        par_table[operator] = UB1_UB2;
        break;

case EQBYT:
case LEBYT:
case GEBYT:

        par_table[operator] = UB1_UB2_B;
        break;

case UJP:
case FJP:
case EFJ:
case NFJ:
case TJP:

        par_table[operator] = SBP;
        break;

case CIP:

        par_table[operator] = DB_UB;
        break;

case CIX:
```

```
                        par_table[operator] = UB1_DB_UB2;
                        break;

                case LSL:

                        par_table[operator] = DBP;
                        break;
                };
        }/*loop*/
        ;
}/*set_ptable*/

get_big()
{
        B = store.code[PC];
        PC = PC+1;
        if(B > 127)
        {
                B = (B-128)*256 + store.code[PC];
                PC = PC+1;
        };
}
```

```
/* main instruction fetch */
fetch()
{
        old_PC = PC;
        operator = store.func[PC];
        PC += 1;
        switch (par_table[operator]) {
        case NONE:
                break;

        case UBP:
                {
                        UB = store.code[PC];
                        PC += 1;
                        break;
                };

        case WP:
                {
                        W = store.code[PC] + store.code[PC+1]*256;
                        PC += 2;
                        break;
                };

        case BP:
                get_big();
                break;

        case DB_B:
                {
                        DB = store.code[PC];
                        PC += 1;
                        get_big();
                        break;
                };

        case UB_B:
                {
                        UB = store.code[PC];
                        PC += 1;
                        get_big();
                        break;
                };

        case UB1_B_UB2:
                {
                        UB1 = store.code[PC];
                        PC += 1;
                        get_big();
                        UB2 = store.code[PC];
                        PC += 1;
                        break;
                };
```

```
        case UB1_UB2:
                {
                        UB1 = store.code[PC];
                        PC += 1;
                        UB2 = store.code[PC];
                        PC += 1;
                        break;
                };

        case UB1_UB2_B:
                {
                        UB1 = store.code[PC];
                        PC += 1;
                        UB2 = store.code[PC];
                        PC += 1;
                        get_big();
                        break;
                };

        case SBP:
                {
                        SB = store.code[PC];
                        PC += 1;
                        if( SB > 127)  SB = SB-256;
                        break;
                };

        case DB_UB:
                {
                        DB = store.code[PC];
                        PC += 1;
                        UB = store.code[PC];
                        PC += 1;
                        break;
                };

        case UB1_DB_UB2:
                {
                        UB1 = store.code[PC];
                        PC += 1;
                        DB  = store.code[PC];
                        PC += 1;
                        UB2 = store.code[PC];
                        PC += 1;
                        break;
                };

        case DBP:
                {
                        DB = store.code[PC];
                        PC += 1;
                        break;
                };
```

```
        case NOT_IMP:
                error = UNIMP;
                break;
        default:
                error = UNIMP;
                break;
        };/*CASE*/
} /*setup_pars*/
```

```
/*General memory management*/

contents(a)
ADDRESS a;
{
        return (store.data[a]);
};

update(a,b)
ADDRESS a;
short b;
{
        store.data[a] = b;
};

/* The stack is declared global to the whole interpreter program, */
/* and is a stack of integers                                     */
st_clear(n)
short n;
{
        ADDRESS ltsp;
        ltsp = SP-proc_slp;
        ltsp = ltsp-n;
        if (ltsp < store.data[CTASK+tlo_stack])
        {
                store.data[sys_base+flt_rec] = EREC;
                store.data[sys_base+flt_wds] = n+proc_slp;
                store.data[sys_base+flt_num] = stack_fault / 2;
                store.data[sys_base+flt_tib] = CTASK;
                error = stack_fault;
                return(FALSE);
        }
        else return(TRUE);
};

pop_mem (x)
ADDRESS x;
{
        store.data[x] = store.data[SP];
        SP = SP+1;
};

pop (x)
short *x;
{
        if (!sex_compatible())
        {
                swap1.sint = store.data[SP];
                swap2.schar[0] = swap1.schar[1];
                swap2.schar[1] = swap1.schar[0];
                *x = swap2.sint;
        }
        else *x = store.data[SP];
        SP = SP+1;
```

```
};

push (x)
short x;
{
        SP = SP-1;
        store.data[SP] = x;
};

peep (x)
short *x;
{
        *x = store.data[SP];
};

pushes (length,a)
short length;
ADDRESS a;
{
        ADDRESS la;
        la = a+length-1;
        if(st_clear(length))
                while (la >= a)
                {
                        push(contents(la));
                        la = la-1;
                };
};

swap_pushes (length,a)
short length;
ADDRESS a;
{
        ADDRESS la;
        short   temp;
        union pun_byte x;
        la = a+length-1;
        if(st_clear(length))
        {
                while (la >= a)
                {
                        x.byte_intf = contents(la);
                        temp = x.bytf[h_order];
                        x.bytf[h_order] = x.bytf[l_order];
                        x.bytf[l_order] = temp;
                        push(x.byte_intf);
                        la = la-1;
                };
        };
};

popes (length,a)
short length;
ADDRESS a;
```

```
{
        ADDRESS lim_address;
        lim_address = a+length-1;
        while (a <= lim_address)
        {
                pop_mem(a);
                a = a+1;
        };
};

pop_address(a)
ADDRESS *a;
{
        union pun_address p;
        short n;
        pop(&n);
        p.address_intf = n;
        *a = p.addf;
};

push_address(a)
ADDRESS a;
{
        union pun_address p;
        short         n;
        p.addf = a;
        n = p.address_intf;
        push(n);
};

pop_real(x)
float *x;
{
        union pun_real p;
        short y1,y2;
        pop(&y1);
        p.real_intf[1] = y1;
        pop(&y2);
        p.real_intf[0] = y2;
        *x = p.realf;
};

push_real(x)
float x;
{
        union pun_real p;
        short y1,y2;
        p.realf = x;
        y1 = p.real_intf[0];
        push(y1);
        y2 = p.real_intf[1];
        push(y2);
};
```

```
pop_bool(b)
short *b;
{
        union pun_bool p;
        short g;
        pop(&g);
        p.bool_intf = g;
        *b = p.boolf;
};

push_bool(b)
short b;
{
        union pun_bool p;
        p.bool_intf = 0;
        p.boolf = b;
        push(p.bool_intf);
};

pop_badr(a)
BYTE_ADDRESS *a;
{
        BYTE_ADDRESS a1;
        pop(&a);
        pop(&a1);
        *a = 2*a1+(*a);
};

/*
push_badr(a)
short a;
{
  push(a / 2);
  push(a % 2);
};
*/

pop_char(ch)
char *ch;
{
        union pun_char p;
        pop(&p.char_intf);
        *ch = p.charf;
};

push_char(ch)
char ch;
{
        union pun_char p;
        p.charf = ch;
        push(p.char_intf);
};

extend(n)
```

```
short n;
{
        SP = SP-n;
};

contract(n)
short n;
{
        SP = SP+n;
        /*stack check should occur here*/
};

/* when the byte address is even then the left hand byte, otherwise */
/* the right hand byte                                               */

cont_bytes (a)
BYTE_ADDRESS a;
{
        return(store.code[a]);
};

upd_bytes (a,b)
BYTE_ADDRESS a;
code_type b;
{
        store.code[a] = b;
};

byte_swap(x)
short *x;
{
        union pun_byte p;
        short temp;
        p.byte_intf = *x;
        temp = p.bytf[0];
        p.bytf[0] = p.bytf[1];
        p.bytf[1] = temp;
        *x = p.byte_intf;
};

sw_bytes (a)
ADDRESS a;
{
        short temp;
        BYTE_ADDRESS x,y;
        x = a*2;
        y = x+1;
        temp = store.code[x];
        store.code[x] = store.code[y];
        store.code[y] = temp;
};

/*moving multiple words or multiple bytes*/
move_words (n,a,b)
```

```
short n;
ADDRESS a,b;
{
        ADDRESS limit;
        limit = a+n-1;
        while (a<=limit)
        {
                store.data[b] = store.data[a];
                a = a+1;
                b = b+1;
        };
};

move_bytes (n,a,b)
short n;
BYTE_ADDRESS a,b;
{
        BYTE_ADDRESS limit;
        limit = a+n-1;
        while (a<=limit)
        {
                store.code[b] = store.code[a];
                a = a+1;
                b = b+1;
        }
};

/*Other pointer descriptors into memory, and addressing mechanisms*/
move_swapping(n,a,b)
short n;
ADDRESS a,b;
{
        ADDRESS limit;
        limit = a+n-1;
        while (a<=limit)
        {
                update(b, contents(a));
                sw_bytes(b);
                a = a+1;
                b = b+1;
        }
};
```

```
c_address(n)
short n;
{
        union pun_address p;
        p.address_intf = n;
        return(p.addf);
};

c_byte_address(n)
short n;
{
        union pun_byddr p;
        p.byddr_intf = n;
        return(p.badf);
};

intermediate_offset (a,offset)
ADDRESS a;
short offset;
{
        return(a+c_address(offset)+disp0);
};

L_offset(offset)
short offset;
{
        return(L+c_address(offset));
};

ext_offset (g, offset)
short g,offset;
{
        ADDRESS temp;
        temp =c_address(store.data[EVEC+g]);
        return(c_address(store.data[temp+env_data]) + disp0 +
            c_address(offset));
};

G_offset(offset)
short offset;
{
        return(G+c_address(offset));
};

constant_offset(offset)
short offset;
{
        return(c_address(store.data[seg_bot+seg_coff]) + c_address(offset));
};

link_traverse(DB)
small_type DB;
{
        ADDRESS p;
```

```
        short     i;
        p = last_frame;
        for (i = 1;i <= DB;i++)
                p = store.data[p];
        return(p);
};

sex_compatible()
{
        return(store.data[seg_bot+seg_bsiw]==1);
};
```

```
/*Unravelling an absolute address from a parameter descriptor*/

pdtoabsolute(pd)
ADDRESS pd;
{
        ADDRESS sb;
        if(store.data[pd]==NIL_ptr)
                return(c_address(store.data[pd+1]));
        else
        {
                sb =
                c_address(store.data[store.data[store.data[pd]+env_sib]
                          + seg_base]);
                if(sb==NIL_ptr)
                {
                        flt_EREC = c_address(store.data[pd]);
                        error = seg_fault;
                        return(NIL_ptr);
                }
                else return(sb + c_address(store.data[pd+1]));
        }
};

/*Assume a field descriptor*/
extract_field(a)
ADDRESS *a;
/*extract a field right justified and zero filled*/
{
        unsigned short i,x,y,z;
        union pun_set ps1,ps2;
        pop(&x);
        pop(&y);
        pop_address(&a);
        ps1.set_intf = store.data[*a];
        ps2.set_intf = 0;
        for (i = 0;i <= (y-1);i++)
                if (ps1.setf & (1<<(x+i)))
                        ps2.setf != 1<<i;
        z = ps2.set_intf;
        return(z);
};

/*For use in unsigned compares*/
compare_US(a,b)
short a,b;
{
        if((a>0) && (b<0))
                return(-1);
        else if((a<0) && (b>0))
                return(1);
        else if(a==b)
                return(0);
        else return((a-b) / (abs(a-b)));
};
```

```
/*code segment manipulation procedures*/
get_sb()
{
        if(old_EREC != EREC)
        {
                if(store.data[store.data[EREC+env_sib]+seg_base]
                    != NIL_ptr)
                {
                        sibp = c_address(store.data[EREC+env_sib]);
                        seg_bot = c_address(store.data[sibp+seg_base]);
                        EVEC = c_address(store.data[EREC+e_vect]);
                        store.data[sibp+s_activity] =
                            store.data[sibp+s_activity]+act_val;
                        G = c_address(store.data[EREC+env_data] + disp0);
                }
                else
                {
                        flt_EREC = EREC;
                        EREC = old_EREC;
                        error = seg_fault;
                };
        };
};

get_procbase(proc_no,p)
/* gets the information necessary to set up an activation record*/
code_type proc_no;
struct proc_rec *p;
{
        ADDRESS a;
        /* KSU added sex compatible check against currently loaded segment */
        if(!sex_compatible())
        {
                swap1.sint = store.data[seg_bot];
                swap2.schar[0] = swap1.schar[1];
                swap2.schar[1] = swap1.schar[0];
                swap1.sint =
                store.data[seg_bot + c_address(swap2.sint) - proc_no];
                swap2.schar[0] = swap1.schar[1];
                swap2.schar[1] = swap1.schar[0];
                a = c_address(seg_bot+swap2.sint);
        }
        else a = c_address(seg_bot+
            store.data[seg_bot + c_address(store.data[seg_bot]) - proc_no]);
        printf("procedure %4d0,proc_no);
        p->code_start = c_byte_address((a+1)*2);
        if(!sex_compatible())
        {
                swap1.sint = store.data[a];
                swap2.schar[0] = swap1.schar[1];
                swap2.schar[1] = swap1.schar[0];
                p->data_size = swap2.sint;
                swap1.sint = store.data[a-1];
                swap2.schar[0] = swap1.schar[1];
```

```
                swap2.schar[1] = swap1.schar[0];
                p->code_exit = c_byte_address(swap2.sint);
        }
        else
        {
                p->data_size = store.data[a];
                p->code_exit = c_byte_address(store.data[a-1]);
                ;
        }
};

get_segmentbase(segno)
code_type segno;
/*swaps a segment, or sets an error if the required seg is off memory*/
{
        old_segbot = seg_bot;
        old_EREC = EREC;
        EREC = c_address(store.data[EVEC+segno]);
        /*ADDRESS of new EVEC*/
        get_sb();
};

local_call(proc_no,st_link)
code_type proc_no;
ADDRESS st_link;
{
        get_procbase(proc_no,&proc_details);
        if(st_clear(proc_details.data_size+4))
        {
                extend(proc_details.data_size);
                push(procnumber);
                push_address(EREC);             /*set up activation record*/
                push(PC-seg_bot*2);             /*relativise*/
                push(last_frame);
                push(st_link);
                last_frame = SP;
                procnumber = proc_no;
                L = last_frame+disp0;
                PC = proc_details.code_start;
        };
};

ext_call(proc_no,st_link)
code_type proc_no;
ADDRESS st_link;
{
        ADDRESS temp;
        get_procbase(proc_no,&proc_details);
        if(st_clear(proc_details.data_size+4))
        {
                extend(proc_details.data_size);
                push(procnumber);
                push(old_EREC);
                push(PC-old_segbot*2);
```

```
                push(last_frame);
                push(st_link);
                procnumber = proc_no;
                last_frame = SP;
                L = last_frame + disp0;
                PC = proc_details.code_start;
                store.data[sibp+s_ref_count]
                    = store.data[sibp+s_ref_count]+1;
        }
        else
        {
                temp = old_EREC;
                old_EREC = EREC;
                EREC = temp;
                get_sb();
                store.data[sys_base+flt_rec] = EREC;
        };
} /*external call*/
;

seg_and_proc(seg_no,proc_no,stat_lnk)
code_type seg_no,proc_no;
ADDRESS stat_lnk;
{
        get_segmentbase(seg_no);
        if(error!=seg_fault)
                ext_call(proc_no,stat_lnk);
};
```

```
/*task support procedures*/
priority(tibp)
ADDRESS tibp;
{
        return(store.code[tibp*2+tpri]);
};

queue(queue_ad,ctib)
ADDRESS queue_ad,ctib;
{
        ADDRESS prev_tib,htib;
        prev_tib = NIL_ptr;
        htib = queue_ad;
        if (htib!=NIL_ptr)
                while(store.code[(ctib*2)+tpri] <= store.code[(htib*2)+tpri])
                {
                        prev_tib = htib;
                        htib = c_address(store.data[htib+tail]);
                        if (htib==NIL_ptr)  goto getout;
                };
getout :
        if(prev_tib == NIL_ptr)
        {
                store.data[ctib+tail] = htib;
                return(ctib);
        }
        else
        {
                store.data[prev_tib+tail] = ctib;
                store.data[ctib+tail] = htib;
                return(queue_ad);
        }
};

signal(sem)
ADDRESS sem;
{
        ADDRESS ctib;
        ctib = c_address(store.data[sem+sm_queue]);
        if(ctib==NIL_ptr)
                store.data[sem+sm_count] = store.data[sem+sm_count]+1;
        else
        {
                store.data[sem+sm_queue] = store.data[ctib+tail];
                store.data[ctib+thang] = NIL_ptr;
                store.data[ctib+tail] = NIL_ptr;
                READY_Q = queue(READY_Q,ctib);
                if(priority(READY_Q) >= priority(CTASK))
                        error = task_sw;
        };
};
```

```
save_context()
{
        store.code[CTASK*2+proc_n] = procnumber;
        store.data[CTASK+tsp] = SP;
        store.data[CTASK+tmpd0] = L-disp0;
        store.data[CTASK+terec] = EREC;
        store.data[CTASK+tpc] = PC-(seg_bot)*2;
};

restore_context()
{
        procnumber = store.code[CTASK*2+proc_n];
        SP = store.data[CTASK+tsp];
        L = store.data[CTASK+tmpd0]+disp0;
        last_frame = L-disp0;
        EREC = store.data[CTASK+terec];
        sibp = store.data[EREC+env_sib];
        seg_bot = store.data[sibp+seg_base];
        EVEC = store.data[EREC+e_vect];
        G = store.data[EREC+env_data] + disp0;
        PC = store.data[CTASK+tpc] + seg_bot*2;
};

run_task()
{
        CTASK = READY_Q;
        READY_Q = store.data[READY_Q+tail];
        restore_context();
        if(seg_bot==NIL_ptr)
        {
                flt_EREC = EREC;
                error = seg_fault;
        };
};
```

```
/************************************************************************/
/* This routine is the connection between the UCSD p-System and the    */
/* real world. This is the RSP or Runtime Support Package. It is where */
/* the BIOS or BASIC I/O SUBSYSTEM is called from.                      */
/************************************************************************/

/*procedures called from CGX*/
/*this procedure is relevant when in native mode*/
r_seg()
{
        int x;
        pop(&x);
        pop(&x);
        pop(&x);
        pop(&x);/*segment base*/
        if (store.data[x+re_list] != 0) error = native_mode;
}
/*move instructions*/
mvl()
{
        int x;
        BYTE_ADDRESS a,b;
        pop(&x);
        pop_badr(&b);
        pop_badr(&a);
        move_bytes(x,a,b);
}
mvr()
{
        int x;
        BYTE_ADDRESS a,b;
        int i;
        pop(&x);
        pop_badr(&b);
        pop_badr(&a);
        for (i=x-1;i>=0;i--) upd_bytes(b+i,cont_bytes(a+i));
}
flc()
{
        union pun_byte x;
        int n;
        BYTE_ADDRESS a;
        int i;
        pop(&n);
        x.byte_intf=n;
        pop(&n);
        pop_badr(&a);
        for(i=0;i<=(n-1);i++)
                upd_bytes(a+i,x.bytf[1]);
}
/*scan instruction*/
scn()
{
        BYTE_ADDRESS a;
```

```
        char ch;
        BOOLEAN skip;
        int disp,i;
        contract(1);
        pop_badr(&a);
        pop_char(&ch);
        pop_bool(&skip);
        pop(&disp);
        pop(&i);
        i=0;
        if(skip){
                if(disp<0) while((i>disp)&&(ch==cont_bytes(a+i)))
                        i--;
                else while((i<disp)&&(ch==cont_bytes(a+i))) i++;
        }
        else {
                if (disp<0) while((i>disp)&&(ch!=cont_bytes(a+i))) i--;
                else
                        while ((i<disp)&&(ch!=cont_bytes(a+i))) i++;
        }
}
uread()
{
        /*********************************************************/
        /*            DEVICES and DEVICE NUMBERS                 */
        /*********************************************************/
        /* Unitnumber        Volume Name                         */
        /*     0             <reserved for the system>           */
        /*     1             CONSOLE                             */
        /*     2             SYSTERM                             */
        /*     3             <reserved for the system>           */
        /*     4             disk 0                              */
        /*     5             disk 1                              */
        /*     6             PRINTER                             */
        /*     7             REMIN                               */
        /*     8             REMOUT                              */
        /*     9             disk 2                              */
        /*    10             disk 3                              */
        /*    11             disk 4                              */
        /*    12             disk 5                              */
        /*    13-127         <reserved for future expansion>     */
        /*********************************************************/
        /*********************************************************/
        /* Due to the way C interfaces with the different        */
        /* file descriptors, the above unit number will have     */
        /* to be trapped and connected differently.              */
        /* this will have to be done in all routines which       */
        /* have access to a unit via the unitnumber              */
        /* unitread, unitwrite, unitbusy, etc.                   */
        /* the best way will probably be to open the file or     */
        /* device and then close it as the final statement       */
        /* before leaving the function.                          */
        /*********************************************************/
        int i,b_no,length,u_no,control,IORESULT;
```

```
        BYTE_ADDRESS buf_ad;
        pop(&control);
        pop(&b_no);
        pop(&length);
        pop_badr(&buf_ad);
        pop(&u_no);
        if (u_no==sys_no) printf("uread %7d0,b_no);
        /*insert seek and read here*/
        store.data[sys_base+io_rslt]=IORESULT;
}
uwrite()
{
        int i,b_no,length,u_no,control,IORESULT;
        BYTE_ADDRESS buf_ad;
        pop(&control);
        pop(&b_no);
        pop(&length);
        pop_badr(&buf_ad);
        pop(&u_no);
        /*put seek and write here*/
        store.data[sys_base+io_rslt] = IORESULT;
}
ioc()
{
        /*this instruction not implemented*/
}
ior()
{
        int x;
        pop(&x);
        x=store.data[sys_base+io_rslt];
        push(x);
}
ubusy()
{
        contract(1);
        /*this instruction is not implemented*/
        push_bool(FALSE);
}
uwait()
{
        /*this instruction is not implemented*/
        contract(1);
}
uclear()
{
        int x;
        pop(&x);
        if (( x ==  1 ) || ( x ==  2 ) || ( x ==  4 ) || ( x ==  5 ) ||
            ( x ==  6 ) || ( x ==  7 ) || ( x ==  8 ) || ( x ==  9 ) ||
            ( x == 10 ) || ( x == 11 ) || ( x == 12 ))
                store.data[sys_base+io_rslt] = 0;
}
ustatus()
```

```
{
        error=UNIMP;
}
pot()
{
        int n,i;
        float x;
        pop(&n);
        pop_real(&x);
        if(( x < 0 ) || ( x > 38 ) || ( n < 0 ))
        {
                push_real(0);
                error=fpierr;
        }
        else {
                x = 1;
                for ( i = 1; i <= n; i++) x = x * 10;
                push_real(x);
        }
}
/*id search*/
ids(){
        BYTE_ADDRESS textln_prt, cursor_ptr, t_p, t_l;
        int i,j,hash;
        BOOLEAN end_id;
        union pun_char p;
        char ch;
        pop(&textln_ptr);
        pop(&cursor_ptr);
        t_p = textln_ptr*2 + contents (cursor_ptr);
        t_l = (cursor_ptr+3)*2;
        end_id = FALSE;
        for ( i = 0; i <= 7; i++) store.fchar[t_l+i] = ' ';
        i = 0;
        while (! end_id){
                p.char_intf = cont_bytes(t_p);
                if((p.charf >= 'a') && (p.charf <= 'z'))
                        p.char_intf = p.char_intf + ( 'A' - 'a' );
                ch = p.charf;
                if ((( ch >= 'A' ) && ( ch <= 'Z')) ||
                    (( ch >= '0' ) && ( ch <= '9' )))
                {
                        if (i <= 7 ) store.fchar[t_l+i] = ch;
                        i++;
                        t_p++;
                }
                else if ( ch != '_' ) end_id = TRUE;
                else t_p++;
        }
        t_p--;
        store.data[cursor_ptr] = t_p - textln_ptr * 2;
        hash = res_tbl[store.code[t_l] - 'A'];
        hash--;
        do {
```

```
                hash++;
                i=0;
                while  ((search_table[hash].word[i+1]==store.fchar[t_l+i]) &&
                        (i<7))
                        i++;
        }
        while((i!=7) && (i!=0));
        if ( i == 0 ) {
        }
        else {
                if (search_table[hash].word[i+1] == store.fchar[t_l+i])
                {
                        store.data[cursor_ptr+1] = search_table[hash].token[1];
                        store.data[cursor_ptr+2] = search_table[hash].token[2];
                }
        }
}
trs()
{
        BYTE_ADDRESS root_ptr,found_ptr,target_ptr;
        BOOLEAN search;
        int throw_away,i;
        pop(&target_ptr);
        target_ptr = target_ptr * 2;
        pop(&found_ptr);
        pop(&root_ptr);
        root_ptr = root_ptr * 2;
        pop(&throw_away);
        while (search){
                i = 0;
                while (( i <= 7 ) &&
                       ( store.code[target_ptr + i] ==
                         store.code[root_ptr + i]))
                        i++;
                if ( i == 8 ) {
                        push(0);
                        search = FALSE;
                }
                else {
                        if(store.code[root_ptr+i] < store.code[target_ptr+i])
                        {
                                if (store.data[(root_ptr + 8) / 2] != NIL_ptr)
                                        root_ptr =
                                        store.data[(root_ptr + 8) / 2] * 2;
                                else {
                                        push(1);
                                        search = FALSE;
                                }
                        }
                        else {
                                if (store.data[(root_ptr + 10) / 2] != NIL_ptr)
                                        root_ptr =
                                        store.data[(root_ptr + 10) / 2] * 2;
                                else {
```

```
                                        push(-1);
                                        search = FALSE;
                                }
                        }
                }
        }
        store.data[found_ptr] = root_ptr / 2;
}
attach() {
        error = UNIMP;
}
tim(){
        ADDRESS x;
        pop(&x);
        store.data[x] = 0;
        pop(&x);
        store.data[x] = 0;
        /*this instruction is unimplemented since it is */
        /*implementation dependent                      */
}
sp_rtns(proc_no,t)
int proc_no;
int *t;
{
        *t=TRUE;
        switch (proc_no)
        {
        case 4:
                {
                        r_seg();
                        break;
                }
        case 5:
        case 6:
        case 7:
        case 8:
        case 9:
        case 10:
        case 11:
        case 12:
        case 13:
        case 14:
                {
                        *t=FALSE;
                        break;
                }
        case 15:
                {
                        mvl();
                        break;
                }
        case 16:
                {
                        mvr();
```

```
                    break;
                }
        case 17:
                {
                    *t=FALSE;
                    break;
                }
        case 18:
                {
                    uread();
                    break;
                }
        case 19:
                {
                    uwrite();
                    break;
                }
        case 20:
                {
                    tim();
                    break;
                }
        case 21:
                {
                    flc();
                    break;
                }
        case 22:
                {
                    scn();
                    break;
                }
        case 23:
                {
                    ioc();
                    break;
                }
        case 24:
        case 25:
        case 26:
        case 27:
        case 28:
                {
                    *t=FALSE;
                    break;
                }
        case 29:
                {
                    attach();
                    break;
                }
        case 30:
                {
                    ior();
```

```
                        break;
                }
        case 31:
                {
                        ubusy();
                        break;
                }
        case 32:
                {
                        pot();
                        break;
                }
        case 33:
                {
                        uwait();
                        break;
                }
        case 34:
                {
                        uclear();
                        break;
                }
        case 35:
                {
                        *t=FALSE;
                        break;
                }
        case 36:
                {
                        ustatus();
                        break;
                }
        case 37:
                {
                        ids();
                        break;
                }
        case 38:
                {
                        trs();
                        break;
                }
        default:
                {
                        *t=FALSE;
                        break;
                }
        }
}

/*special global external call routine*/
glob_ext_call(seg_no,proc_no)
code_type seg_no,proc_no;
{
```

```
        BOOLEAN tr;
        tr = FALSE;
        if (seg_no==1)
                sp_rtns(proc_no,&tr);
        if (!tr)
                seg_and_proc(seg_no,proc_no,G-disp0);
};
```

```
UNION(a,b)
unsigned short a,b;
{
        union pun_set p1,p2;
        p1.set_intf = a;
        p2.set_intf = b;
        p1.setf = p1.setf+p2.setf;
        return(p1.set_intf);
};

intersection(a,b)
unsigned short a,b;
{
        union pun_set p1,p2;
        p1.set_intf = a;
        p2.set_intf = b;
        p1.setf = p1.setf * p2.setf;
        return(p1.set_intf);
};

difference(a,b)
unsigned short a,b;
{
        union pun_set p1,p2;
        p1.set_intf = a;
        p2.set_intf = b;
        p1.setf = p1.setf-p2.setf;
        return(p1.set_intf);
};

equalset(a,b)
unsigned short a,b;
{
        union pun_set p1,p2;
        p1.set_intf = a;
        p2.set_intf = b;
        return(p1.setf==p2.setf);
};

leset(a,b)
unsigned short a,b;
{
        union pun_set p1,p2;
        p1.set_intf = a;
        p2.set_intf = b;
        return(p1.setf<=p2.setf);
};

geset(a,b)
unsigned short a,b;
{
        union pun_set p1,p2;
        p1.set_intf = a;
        p2.set_intf = b;
```

```
        return(p1.setf>=p2.setf);
};

word(a)
BYTE_ADDRESS a;
{
        return((a%2)==0);
};

min(x,y)
short x,y;
{
        if (x<y)
                return(x);
        else return(y);
};

max(x,y)
short x,y;
{
        if (x>y)
                return(x);
        else return(y);
};

/*set operations*/
gen_setop(x,y,a1,a2)
short *x,*y;
ADDRESS *a1,*a2;
{
        pop(&x);
        *a1 = SP;
        contract(x);
        pop(&y);
        *a2 = SP;
        contract(y);
};
```

```
set_adjust()
{
        unsigned short x,y;
        ADDRESS a;
        short i;
        pop(&x);
        y = UB-x;
        if (y>0)
        {
                if (st_clear(y))
                {
                        a = SP-y;
                        move_bytes(2*x,SP*2,a*2);
                        SP = a;
                        for(i=0;i<=(y*2-1);i++)
                                upd_bytes((2*(a+x))+i,0);
                }
                else push(x);
        }
        else
        {
                a = SP-y;
                for(i=(2*UB)-1;i>=0;i--)
                        upd_bytes(2*a+i, cont_bytes(2*SP+i));
                SP = a;
        };
};

subrange_set()
{
        union pun_set p1;
        unsigned short x,y,i,j;
        /*Build a subrange set*/
        pop(&y);
        pop(&x);
        if (((0<=x) && (x<=4079)) && ((0<=y) && (y<=4079)))
        {
                if (x<=y)
                {
                        p1.set_intf = 0;
                        for(i=0;i<=(y % 16);i++)
                                p1.setf != 1<<i;
                        i = p1.set_intf;
                        push(i);
                        j = y / 16;
                        for(i=(x / 16)+1;i<=j;i++)
                                push(-1);
                        pop(&i);
                        p1.set_intf = i;
                        for(j=0;j<=((x % 16)-1);j++)
                                p1.setf &= ~(1<<j);
                        i = p1.set_intf;
                        push(i);
                        for(i=1;i<=(x / 16);i++)
```

```
                                push(0);
                        j = y / 16+1;
                        push(j);
                }
                else push(0);
        }
        else error = inv_ndx;
};

set_inclusion()
{
        union pun_set p1;
        unsigned short x,y,z,wp,i;
        pop(&x);
        i = SP;
        contract(x);
        pop(&y);
        wp = y / 16;
        z = y % 16;
        if (wp>x) push_bool(FALSE);
        else
        {
                p1.set_intf = store.data[wp+i];
                push_bool(p1.setf & (1<<z));
        }
};

set_union()
{
        unsigned short x,y;
        ADDRESS a,a1,a2;
        short i;
        gen_setop(&x,&y,&a1,&a2);
        if (y>=x)  a = a2;
        else a = a1;
        for(i=0;i<=min(x,y)-1;i++)
                store.data[a+i] = UNION(store.data[a1+i],store.data[a2+i]);
        if (x>y)
        {
                a = a1+x;
                a2 = SP;
                for(i=1;i<=x;i++)
                        store.data[a2-i] = store.data[a-i];
        };
        extend(max(x,y));
        push(max(x,y));
};

set_intersection()
{
        unsigned short x,y;
        ADDRESS a,a1,a2;
        short i;
        gen_setop(&x,&y,&a1,&a2);
```

```
        a = a2+y;
        a2 = a2+min(x,y);
        a1 = a1+min(x,y);
        for(i=min(x,y);i<=1;i++)
                store.data[a-i] =
                intersection(store.data[a1-i],store.data[a2-i]);
        extend(min(x,y));
        push(min(x,y));
};

set_difference()
{
        unsigned short x,y;
        ADDRESS a1,a2;
        short i;
        gen_setop(&x,&y,&a1,&a2);
        a1 = a1+min(x,y);
        a2 = a2+min(x,y);
        for(i=min(x,y);i>=1;i--)
        {
                store.data[a2-i] =
                difference(store.data[a2-i],store.data[a1-i]);
        }
        extend(y);
        push(y);
};
```

```
/*Byte comparison procedure*/
byte_compare()
{
        short i,x,y;
        BOOLEAN b1,b2;
        BYTE_ADDRESS a1,a2;
        i = 0;
        b1 = FALSE;
        b2 = FALSE;
        x = 0;
        y = 0;
        pop_address(&a1);
        a1 = 2*a1;
        pop_address(&a2);
        a2 = 2*a2;
        if (UB1!=0)
        {
                a1 = seg_bot*2+a1;
                if (UB1==2) b1 = !sex_compatible();
        };
        if (UB2!=0)
        {
                a2 = seg_bot*2+a2;
                if (UB2==2) b2 = !sex_compatible();
        };
        while ((x==y) && (i<=B))
        {
                if (b1)
                        if (word(a1+i))
                                x = cont_bytes(a1+i+1);
                        else x = cont_bytes(a1+i-1);
                else x = cont_bytes(a1+i);
                if (b2)
                        if (word(a2+i))
                                y = cont_bytes(a2+i+1);
                        else y = cont_bytes(a2+i-1);
                else y = cont_bytes(a2+i);
                i = i+1;
        };
        return(x-y);
};
```

```
/*general string comparison procedure*/
string_compare()
{
        BYTE_ADDRESS a,b;
        short la,lb,i,x,y;
        pop(&a);
        pop(&b);
        a = a*2;
        b = b*2;
        if(UB1!=0)
                a = seg_bot*2 + a;
        if(UB2!=0)
                b = seg_bot*2 + b;
        la = store.code[a];
        lb = store.code[b];
        i = 1;
        x = 0;
        y = 0;
        while ((i<=min(la,lb)) && (x==y))
        {
                x = cont_bytes(a+i);
                y = cont_bytes(b+i);
                i = i+1;
        };
        if(x==y)  return(la-lb);
        else return(x-y);
};
```

```
decode()
{
        unsigned short x,y,z,i;
        BOOLEAN b1;
        ADDRESS a,a1,a2;
        BYTE_ADDRESS ba,ba1,ba2;
        union pun_set p1,p2;
        float  r1,r2;
        switch(operator)
        {
                /*Constant one word loads*/
        case SLDC0:
        case SLDC1:
        case SLDC2:
        case SLDC3:
        case SLDC4:
        case SLDC5:
        case SLDC6:
        case SLDC7:
        case SLDC8:
        case SLDC9:
        case SLDC10:
        case SLDC11:
        case SLDC12:
        case SLDC13:
        case SLDC14:
        case SLDC15:
        case SLDC16:
        case SLDC17:
        case SLDC18:
        case SLDC19:
        case SLDC20:
        case SLDC21:
        case SLDC22:
        case SLDC23:
        case SLDC24:
        case SLDC25:
        case SLDC26:
        case SLDC27:
        case SLDC28:
        case SLDC29:
        case SLDC30:
        case SLDC31:
                push(operator);
                break;
        case LDCN:
                push_address(NIL_ptr);
                break;
        case LDCB:
                push(UB);
                break;
        case LDCI:
                push(W);
                break;
```

```
        case LC0:
                push(constant_offset(B));
                break;

                /*Local one-word loads and stores*/
        case SLDL1:
        case SLDL2:
        case SLDL3:
        case SLDL4:
        case SLDL5:
        case SLDL6:
        case SLDL7:
        case SLDL8:
        case SLDL9:
        case SLDL10:
        case SLDL11:
        case SLDL12:
        case SLDL13:
        case SLDL14:
        case SLDL15:
        case SLDL16:
                push(contents(L_offset(operator-SLDL1+1)));
                break;

        case LDL:
                push(contents(L_offset(B)));
                break;

        case SLLA1:
        case SLLA2:
        case SLLA3:
        case SLLA4:
        case SLLA5:
        case SLLA6:
        case SLLA7:
        case SLLA8:
                push_address(L_offset(operator-SLLA1+1));
                break;

        case LLA:
                push_address(L_offset(B));
                break;

        case SSTL1:
        case SSTL2:
        case SSTL3:
        case SSTL4:
        case SSTL5:
        case SSTL6:
        case SSTL7:
        case SSTL8:
                pop_mem(L_offset(operator-SSTL1+1));
                break;
```

```
        case STL:
                pop_mem(L_offset(B));
                break;

                /*Global one word loads and store*/
        case SLD01:
        case SLD02:
        case SLD03:
        case SLD04:
        case SLD05:
        case SLD06:
        case SLD07:
        case SLD08:
        case SLD09:
        case SLD010:
        case SLD011:
        case SLD012:
        case SLD013:
        case SLD014:
        case SLD015:
        case SLD016:
                /*KSU sex compability check inserted*/
                if(!sex_compatible())
                {
                        swap1.sint = contents(G_offset(operator-SLD01+1));
                        swap2.schar[0] = swap1.schar[1];
                        swap2.schar[1] = swap1.schar[0];
                        push(swap2.sint);
                }
                else push(contents(G_offset(operator-SLD01+1)));
                break;

        case LD0:
                push(contents(G_offset(B)));
                break;

        case LA0:
                push_address(G_offset(B));
                break;

        case SR0:
                pop_mem(G_offset(B));
                break;

                /*Intermediate one-word loads and store*/
        case SL0D1:
        case SL0D2:
                push(contents(intermediate_offset(link_traverse(
                operator-SL0D1+1),B)));
                break;

        case L0D:
                push(contents(intermediate_offset(link_traverse(DB),B)));
                break;
```

```
        case LDA:
                push_address(intermediate_offset(link_traverse(DB),B));
                break;

        case STR:
                pop_mem(intermediate_offset(link_traverse(DB),B));
                break;

                /*Extended one word loads and stores*/
        case LDE:
                push(contents(ext_offset(UB,B)));
                break;

        case LAE:
                push_address(ext_offset(UB,B));
                break;

        case STE:
                pop_mem(ext_offset(UB,B));
                break;

                /*Indirect one-word loads and stores*/
        case SIND0:
        case SIND1:
        case SIND2:
        case SIND3:
        case SIND4:
        case SIND5:
        case SIND6:
        case SIND7:
                {
                        pop_address(&a);
                        push(contents(a+(operator-SIND0)));
                        break;
                };

        case IND:
                {
                        pop_address(&a);
                        push(contents(a+B));
                        break;
                };

        case STO:
                {
                        pop(&x);
                        pop_address(&a);
                        update(a, x);
                        break;
                };

                /*Multiple word loads and stores*/
        case LDC:
```

```
                {
                        x = seg_bot+constant_offset(B);
                        if((UB1==2) && (!sex_compatible()))
                                swap_pushes(UB2,x);
                        else pushes(UB2,x);
                        break;
                };

        case LDM:
                {
                        pop(&x);
                        pushes(UB,x);
                        if(error!=OK)  push(x);
                        break;
                };

        case STM:
                {
                        x = store.data[SP+UB];
                        popes(UB,x);
                        pop(&x);
                        break;
                };

        case LDCRL:
                {
                        push(contents(seg_bot+constant_offset(B+1)));
                        push(contents(seg_bot+constant_offset(B)));
                        break;
                };

        case LDRL:
                {
                        pop(&a);
                        push(contents(a+1));
                        push(contents(a));
                        break;
                };

        case STRL:
                {
                        a = store.data[SP+2];
                        pop_mem(a);
                        pop_mem(a+1);
                        pop(&a);
                        break;
                };

        case EXPRL:
                {
                        pop_real(&r1);
                        push_real(0);
                        push_real(r1);
                        break;
```

```
            };

        case CTRL:
            {
                    pop_real(&r1);
                    pop_real(&r2);
                    push_real(r1);
                    break;
            };

            /*String and packed array of character parameter copying*/
        case CAP:
            {
                    peep(&x);
                    y = pdtoabsolute(x);
                    if(error!=seg_fault)
                    {
                            contract(1);
                            pop(&z);
                            move_words(B, y, z);
                    }
                    break;
            };

        case CSP:
            {
                    peep(&x);
                    y = pdtoabsolute(x);
                    if(error!=seg_fault)
                    {
                            if(store.code[2*y]>UB)
                                    error = string_fault;
                            else
                            {
                                    pop(&z);
                                    pop(&z);
                                    move_bytes(cont_bytes(2*y)+1,2*y,2*z);
                            }
                    }
                    break;
            };

            /*Byte load and store*/
        case LDB:
            {
                    pop_badr(&x);
                    push(cont_bytes(x));
                    break;
            };

        case STB:
            {
                    pop(&x);
                    pop_badr(&y);
```

```
                upd_bytes(y,x);
                break;
        };

        /*Packed field load and store*/
    case LDP:
        push(extract_field(&a));
        break;

    case STP:
        {
                pop(&x);
                peep(&y);
                p1.set_intf = extract_field(&a);
                p2.set_intf = contents(a);
                /*z = 2**y;*/
                z = 1;
                for(i=1;i<=y;i++) z = z*2;
                p1.set_intf = (p1.set_intf)*z;
                p2.set_intf = p2.set_intf-p1.set_intf;
                p2.set_intf = p2.set_intf+(x*z);
                store.data[a] = p2.set_intf;
                break;
        };

        /*Record and array indexing and assignment*/
    case MOV:
        {
                pop(&x);
                pop(&y);
                if(UB!=0)
                {
                        x = seg_bot+x;
                        if((UB==2)&&(!sex_compatible()))
                                move_swapping(B,x,y);
                        else move_words(B,x,y);
                }
                else move_words(B,x,y);
                break;
        };

    case INCR:
        {
                pop(&x);
                push(x+B);
                break;
        };

    case IXA:
        {
                pop(&x);
                pop(&y);
                push(y+B*x);
                break;
```

```
            };

        case IXP:
            {
                    pop(&a);
                    x = a / UB1;
                    y = a % UB1;
                    pop(&z);
                    push(z+x);
                    push(UB2);      /*push packed field descriptor*/
                    push(UB2*y);
                    break;
            };

            /*Logical top of stack operations*/
        case LAND:
            {
                    pop(&x);
                    p1.set_intf = x;
                    pop(&x);
                    p2.set_intf = x;
                    p1.setf = p1.setf*p2.setf;
                    x = p1.set_intf;
                    push(x);
                    break;
            };

        case LOR:
            {
                    pop(&x);
                    p1.set_intf = x;
                    pop(&x);
                    p2.set_intf = x;
                    p1.setf = p1.setf+p2.setf;
                    x = p1.set_intf;
                    push(x);
                    break;
            };

        case LNOT:
            {
                    pop(&x);
                    p1.set_intf = x;
                    p1.setf = (~p1.setf);
                    x = p1.set_intf;
                    push(x);
                    break;
            };

        case BNOT:
            {
                    pop_bool(&b1);
                    if(b1) push(0);
                    else   push(1);
```

```
                        break;
                };

        case LEQU:
                {
                        pop(&x);
                        pop(&y);
                        push_bool(compare_US(x,y)>=0);
                        break;
                };

        case GEQU:
                {
                        pop(&x);
                        pop(&y);
                        push_bool(compare_US(x,y)<=0);
                        break;
                };

                /*Integer top of stack arithmetic*/
        case ABI:
                {
                        pop(&x);
                        push(abs(x));
                        break;
                };

        case NGI:
                {
                        pop(&x);
                        push(-x);
                        break;
                };

        case INCI:
                {
                        pop(&x);
                        push(x+1);
                        break;
                };

        case DECI:
                {
                        pop(&x);
                        push(x-1);
                        break;
                };

        case ADI:
                {
                        pop(&x);
                        pop(&y);
                        push(x+y);
                        break;
```

```
            };

        case SBI:
            {
                    pop(&x);
                    pop(&y);
                    push(y-x);
                    break;
            };

        case MPI:
            {
                    pop(&x);
                    pop(&y);
                    push(x*y);
                    break;
            };

        case DVI:
            {
                    pop(&x);
                    pop(&y);
                    push(y / x);
                    break;
            };

        case MODI:
            {
                    pop(&x);
                    pop(&y);
                    push(y % x);
                    break;
            };

        case CHK:
            {
                    pop(&x);
                    pop(&y);
                    peep(&z);
                    if ((z<y) || (x<z))  error = inv_ndx;
                    break;
            };

        case EQUI:
            {
                    pop(&x);
                    pop(&y);
                    push_bool(x=y);
                    break;
            };

        case NEQI:
            {
                    pop(&x);
```

```
                pop(&y);
                push_bool(x!=y);
                break;
        };

    case LEQI:
        {
                pop(&x);
                pop(&y);
                push_bool(y<=x);
                break;
        };

    case GEQI:
        {
                pop(&x);
                pop(&y);
                push_bool(y>=x);
                break;
        };

        /*Real top of stack arithmetic- integers of 16 bits*/
    case FLT:
        {
                pop(&x);
                r1 = x;
                push_real(r1);
                break;
        };

    case TNC:
        {
                pop_real(&r1);
                push(r1);
                break;
        };

    case RND:
        {
                pop_real(&r1);
                push(r1);
                break;
        };

    case ABR:
        {
                pop_real(&r1);
                push_real(fabs(r1));
                break;
        };

    case NGR:
        {
                pop_real(&r1);
```

```
                push_real(-r1);
                break;
        };

    case ADR:
        {
                pop_real(&r1);
                pop_real(&r2);
                push_real(r1+r2);
                break;
        };

    case SBR:
        {
                pop_real(&r1);
                pop_real(&r2);
                push_real(r2-r1);
                break;
        };

    case MPR:
        {
                pop_real(&r1);
                pop_real(&r2);
                push_real(r1*r2);
                break;
        };

    case DVR:
        {
                pop_real(&r1);
                pop_real(&r2);
                push_real(r2/r1);
                break;
        };

    case EQREAL:
        {
                pop_real(&r1);
                pop_real(&r2);
                push_bool(r1=r2);
                break;
        };

    case LEREAL:
        {
                pop_real(&r1);
                pop_real(&r2);
                push_bool(r2<=r1);
                break;
        };

    case GEREAL:
        {
```

```
                pop_real(&r1);
                pop_real(&r2);
                push_bool(r2>=r1);
                break;
        };

        /*Set top of stack operations*/
case ADJ:
        set_adjust();
        break;

case SRS:
        subrange_set();
        break;

case INN:
        set_inclusion();
        break;

case UNI:
        set_union();
        break;

case INT:
        set_intersection();
        break;

case DIF:
        set_difference();
        break;

case EQPWR:
        {
                b1 = TRUE;
                gen_setop(&x,&y,&a1,&a2);
                z = min(x,y);
                for(i=0;i<=z-1;i++)
                        b1 =
                        (b1 &&
                        (equalset(store.data[a1+i],store.data[a2+i])));
                if (x>y) for(i=y;i<=x-1;i++)
                        b1 = b1 && (store.data[a1+i]=0);
                else if (x<y) for(i=x;i<=y-1;i++)
                        b1 = (b1 && (store.data[a2+i]=0));
                push_bool(b1);
                break;
        };

case LEPWR:
        {
                b1 = TRUE;
                gen_setop(&x,&y,&a1,&a2);
                z = min(x,y);
                for(i=0;i<=z-1;i++)
```

```
                        b1 =
                        (b1 &&
                        (leset(store.data[a2+i],store.data[a1+i])));
                if (x<y) for(i=x;i<=y-1;i++)
                        b1 = (b1 && (store.data[a2+i]=0));
                push_bool(b1);
                break;
        };

case GEPWR:
        {
                b1 = TRUE;
                gen_setop(&x,&y,&a1,&a2);
                z = min(x,y);
                for(i=0;i<=z-1;i++)
                        b1 =
                        (b1 &&
                        (geset(store.data[a2+i],store.data[a1+i])));
                if (x>y) for(i=y;i<=x-1;i++)
                        b1 = (b1 && (store.data[a1+i]=0));
                push_bool(b1);
                break;
        };

        /*Byte array comparisons*/
case EQBYT:
        {
                push_bool(byte_compare()==0);
                break;
        };

case LEBYT:
        {
                push_bool(byte_compare()>=0);
                break;
        };

case GEBYT:
        {
                push_bool(byte_compare()<=0);
                break;
        };

        /*jumps*/
case UJP:
        PC = PC+SB;
        break;

case FJP:
        {
                pop_bool(&b1);
                if (!b1) PC = PC+SB;
                break;
        };
```

```
        case TJP:
                {
                        pop_bool(&b1);
                        if (b1) PC = PC+SB;
                        break;
                };

        case EFJ:
                {
                        pop(&x);
                        pop(&y);
                        if (x!=y) PC = PC+SB;
                        break;
                };

        case NFJ:
                {
                        pop(&x);
                        pop(&y);
                        if (x==y) PC = PC+SB;
                        break;
                };

        case UJPL:
                PC = PC+W;
                break;

        case FJPL:
                {
                        pop_bool(&b1);
                        if (!b1) PC = PC+W;
                        break;
                };

        case XJP:
                {
                        a = seg_bot+constant_offset(B);
                        b1 = !sex_compatible();
                        x = contents(a);      /*minimum index*/
                        y = contents(a+1);    /*maximum index*/
                        if (b1)
                        {
                                byte_swap(&x);
                                byte_swap(&y);
                        };
                        pop(&z);
                        if ((x<=z) && (z<=y))
                        {
                                z = contents(a+2+(z-x));
                                if (b1) byte_swap(&z);
                                PC = PC+z;
                        };
                        break;
```

```
        };

        /*Procedure and function calls and returns*/
    case CLP:
        local_call(UB,last_frame);
        break;

    case CGP:
        local_call(UB,G-disp0);
        break;

    case SCIP1:
    case SCIP2:
        local_call(UB,link_traverse((operator)-(SCIP1)+1));
        break;

    case CIP:
        local_call(UB,link_traverse(DB));
        break;

    case CLX:
        seg_and_proc(UB1,UB2,last_frame);
        break;

    case SCGX1:
    case SCGX2:
    case SCGX3:
    case SCGX4:
    case SCGX5:
    case SCGX6:
    case SCGX7:
    case SCGX8:
        glob_ext_call(operator-SCGX1+1,UB);
        break;

    case CGX:
        glob_ext_call(UB1,UB2);
        break;

    case CIX:
        seg_and_proc(UB1,UB2,link_traverse(DB));
        break;

    case CFP:
        {
            pop(&x);
            old_segbot = seg_bot;
            old_EREC = EREC;
            pop(&EREC);
            get_sb();
            if (error != seg_fault)
            {
                pop(&a);
                ext_call(x,a);
```

```
                }
                break;
        };

    case RPU:
        {
                old_EREC = EREC;
                old_segbot = seg_bot;
                EREC = store.data[L-1];        /*EREC of calling proc*/
                get_sb();
                if (error!=seg_fault)
                {
                        if (old_EREC!=EREC)
                                store.data[sibp+s_ref_count]
                                    = store.data[sibp+s_ref_count]-1;
                        SP = L-disp0;
                        contract(1);
                        pop(&last_frame);
                        L = last_frame+disp0;
                        pop(&PC);
                        contract(1);           /*EREC link*/
                        pop(&procnumber);
                        if (procnumber<0)
                        {
                                procnumber = -procnumber;
                                get_procbase(procnumber,&proc_details);
                                PC = seg_bot*2+proc_details.code_exit;
                        }
                        else PC = PC+seg_bot*2;
                        contract(B);
                };
                break;
        };

    case LSL:
        push(link_traverse(DB));
        break;

    case BPT:
        error = UNIMP;
        break;

        /*Concurrency support*/
    case SIG:
        {
                pop(&a);                /*semaphore*/
                signal(a);
                break;
        };

    case WAT:
        {
                pop(&a);                /*semaphore*/
                if (store.data[a+sm_count] != 0)
```

```
                        store.data[a+sm_count] =
                        store.data[a+sm_count] - 1;
                else
                {
                        save_context();
                        store.data[CTASK+thang] = a;
                        store.data[a+sm_queue] =
                            queue(store.data[a+sm_queue], CTASK);
                        run_task();
                };
                break;
        };

        /*string instructions*/
case EQSTR:
        {
                push_bool(string_compare()==0);
                break;
        };

case LESTR:
        {
                push_bool(string_compare()>=0);
                break;
        };

case GESTR:
        {
                push_bool(string_compare()<=0);
                break;
        };

case ASTR:
        {
                pop(&a1);
                ba1 = a1*2;
                pop(&a2);
                ba2 = a2*2;
                if (UB1 != 0)
                        ba1 = ba1 + 2*seg_bot;
                if (cont_bytes(ba1) <= UB2)
                {
                        upd_bytes(ba2,cont_bytes(ba1));
                        move_bytes(cont_bytes(ba1),ba1+1,ba2+1);
                }
                else error = string_fault;
                break;
        };

case CSTR:
        {
                pop(&x);
                peep(&a);
                ba = 2*a;
```

```
                        if ((x>cont_bytes(ba)) || (x<=0))
                        {
                                error = inv_ndx;
                                push(1);
                        }
                        else extend(1);
                        break;
                };

                /*miscellaneous instructions*/
        case LPR:
                {
                        pop(&x);
                        if (x<0)
                        {
                                if (x > -4)
                                        switch (x)
                                        {
                                                case (-1) :
                                                push(CTASK);
                                                break;
                                                case (-2) :
                                                push(EVEC);
                                                break;
                                                case (-3) :
                                                push(READY_Q);
                                                break;
                                        };
                        }
                        else
                        {
                                save_context();
                                push(contents(CTASK+x));
                        };
                        break;
                };

        case SPR:
                {
                        pop(&x);
                        pop(&y);          /*value and register number*/
                        save_context();
                        if (y<0)
                        {
                                if (y > (-4))
                                        switch (y)
                                        {
                                                case (-1) :
                                                CTASK = x;
                                                break;
                                                case (-2) :
                                                EVEC = x;
                                                break;
                                                case (-3) :
```

```
                                        READY_Q = x;
                                        break;
                                };
                }
                else store.data[CTASK+y] = x;
                restore_context();
                break;
        };

case DUP1:
        {
                peep(&x);
                push(x);
                break;
        };

case DUP2:
        {
                pop(&x);
                pop(&y);
                extend(2);
                push(y);
                push(x);
                break;
        };

case SWAP:
        {
                pop(&x);
                pop(&y);
                push(x);
                push(y);
                break;
        };

case NOP:
        break;

case NAT:
        error = UNIMP;
        break;

case NAT_INFO:
        PC = PC + B;
        break;

case UNIM1:
case UNIM2:
case UNIM3:
case UNIM4:
case UNIM5:
case UNIM6:
case UNIM7:
case UNIM8:
```

```
        case UNIM9:
        case UNIM10:
        case UNIM11:
        case UNIM12:
        case UNIM13:
        case UNIM14:
        case UNIM15:
        case UNIM16:
        case UNIM17:
        case UNIM18:
        case UNIM19:
        case UNIM20:
        case UNIM21:
        case UNIM22:
        case UNIM23:
        case UNIM24:
        case UNIM25:
        case UNIM26:
        case UNIM27:
        case UNIM28:
        case UNIM29:
        case UNIM30:
        case UNIM31:
        case UNIM32:
        case UNIM59:
        case UNIM62:

        case UNIM73:
        case UNIM74:
        case UNIM75:
        case UNIM76:
        case UNIM77:
        case UNIM78:
        case UNIM79:
        case UNIM80:
                error = UNIMP;
        };/*CASE*/
};/*DECODE*/
```

PORTING THE UCSD p-SYSTEM TO UNIX

by

CARLOS LYNN QUALLS

B. S., University of Arkansas, 1975

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1983

ABSTRACT

This report is a description of an effort to modify the University of Califorina, San Diego P-system operating system to run under the UNIX operating system. This report contains a description of the organization and architecture of the UCSD P-system and P-machine and details on how the P-machine would be emulated under UNIX. It also contains a description of the problems faced in the emulation and how those difficulties were resolved.