Bell Laboratories

Cover Sheet for Technical Memorandu

Title- Emulation of UNIX on Peripheral Processors

Date- January 9, 1975

TM- 75-1352-2

Other Keywords- Minicomputer Support
Multiprocessing

Author(s)	Location and Room	Extension
Lycklama, H.	MH 7C-211	6170
Christensen, C.	MH 7C-217	4441

Charging Case- 39394

Filing Case- 39394-11

ABSTRACT

The UNIX operating system has been emulated on a peripheral PDP-11 computer which has a communication link to a central PDP-11/45 computer running UNIX. Emulation is achieved by passing all traps that cannot be handled by the peripheral processor (PP) to the central processor (CP). This technique enables one to run object code produced by the C, LIL and Fortran compilers, as well as the standard assembler, on the peripheral processor, providing a powerful way of developing software for the PP and of running programs on the PP. The PP has complete access to the file system on the CP, yet the PP does not require a resident UNIX operating system.

This UNIX emulation technique also provides the capability to support a stand-alone PDP-11 minicomputer by connecting it to a CP running UNIX. When the program for the PP is developed and debugged, the link to the CP may be severed, producing a stand-alone system.

Besides providing programming support for a PDP-11 minicomputer, the emulation package also provides the ability to configure a cost-effective multi-processor UNIX system. For example, a minimally configured PDP-11/45 PP may be linked to a central PDP-11/45 processor to run compute-bound programs.

The minimum configuration for any PP is a 4K PDP-11 machine with a communication link to the CP. The entire communication package and trap handler in the PP require only 400 words of code.

Pages Text 20 Other 3 Total 23

No. Figures 0 No. Tables 0 No. Refs. 8

Address Label

DISTRIBUTION
(REFER GEI 13.9-3)

COMPLETE MEMORANDUM TO

CORRESPONDENCE FILES

OFFICIAL FILE COPY
PLUS ONE COPY FOR
EACH ADDITIONAL FILING
CASE REFERENCED

DATE FILE COPY
(FORM E-1328)

10 REFERENCE COPIES

ACKERMAN,A F
ALLES,HAROLD G
ANDERSON,ROBERT V
ANDERSON,WILLIAM A
+ARDIS,R B
BAYER,DOUGLAS L
BEYER,JEAN-DAVID
BILINSKI,D J
BOYD,GARY D
BRAINARD,RALPH C
BREECE,HARRY T III
BREWSTER,HAROLD O
BROWN,W STANLEY
BUCHSBAUM,S J
CAMLET,J V JR
CANADAY,RUDD H
CHRISTENSEN,C
+CLOGSTON,A M
+CONDON,J H
CONNOLLY,C V
CUTLER,C CHAPIN
DOLAN,MRS MARIE T
DOLOTTA,T A
FAULKNER,R A
FISCHER,W C
FREEMAN,K GLENN
+FREENY,S L
GELLIS,H S
+GILLETTE,DEAN
GIORDANO,PHILIP P
GLASSER,ALAN L
GOGUEN,MS NANCY
GRAVEMAN,R F
HAGELBARGER,D W
HAIGHT,R C
HAMMING,R W
+HANNAY,N B
HAUSE,A D
IVIE,EVAN L
JARVIS,JOHN F
JUDICE,CHARLES N
KAISER,J F
KAMINSKI,WILLIAM
KEEFAUVER,W L
KNUDSEN,DONALD B
KUBIK,P S
LICWINKO,J S
LIMB,J O
LOZIER,JOHN C
LUDERER,GOTTFRIED W R
LUDWIG,J J
LYCKLAMA,HEINZ
LYONS,T G
+MALTHANER,W A
MANCUSI,M D
MARANZANO,JOSEPH F
MASHEY,JOHN R
MC ILROY,M DOUGLAS
MCDONALD,H S
MENNINGER,R E
METAXIDES,A
METZLER,MRS HELEN M
MILLER,S E
MOLLENAUER,J F
MORGAN,S P
MUENZER,T B
+NINKE,WILLIAM H
+OSSANNA,J F JR
+PATEL,C K N
PINSON,ELLIOT N
PLAUGER,P J
+PRIM,ROBERT C
ROBERTS,CHARLES S
ROCHKIND,M J
ROCHKIND,M M
RONKOVITZ,FRANK J JR
ROOME,WILLIAM D
ROSENFELD,PETER E
ROWLINSON,D E
SABSEVITZ,A L
SATZ,L R
SJURSEN,C A
+SLICHTER,W P
SMITH,D W
STEVENSON,H P
SWANSON,GEORGE K
SWARTZWELDER,JOHN C
TAGUE,BERKLEY A
TERRY,M E
TEWKSBURY,S K
THOMPSON,BERNARD E
THOMPSON,JOHN S
+THOMPSON,K
TILLOTSON,L C
VAN LAAR,MRS A
WEHR,L A
WELLER,DAVID R
WILD,J CHRISTIAN
WOLONTIS,V MICHAEL
YAMIN,MRS E E
+YOUNG,JAMES A
101 NAMES

COVER SHEET ONLY TO

CORRESPONDENCE FILES

4 COPIES PLUS ONE
COPY FOR EACH FILING
CASE

COVER SHEET ONLY TO

ABRAHAM,STUART A
AHO,A V
AHRENS,RAINER B
ALBERTS,BARBARA A
ALCALAY,DAVID
ALLEN,JAMES R
ALMQUIST,R P
AMORY,R W
AMOSS,JOHN J
ANDERSON,M M
ANDERSON,MRS C M
ARNOLD,GEORGE W
ARNOLD,S L
ARTHURS,EDWARD
ATAL,B S
AVERILL,R M JR
BAKER,B S
BALDWIN,G L
BALDWIN,GARY L
BARBER,A J
BARTLETT,WADE S
BASEIL,RICHARD J
BAUER,BARBARA T
BAUGH,C R
BECKETT,J T
BENGTSON,A H
BENJAMIN,O CONNELL J
BERGLAND,G DAVID
BERNSTEIN,LAWRENCE
BERRANG,J E
BIAZZO,MARTIN R
BIGELOW,J H JR
BILOWOS,RICHARD M
BIRCHALL,R H
BIREN,MRS IRMA B
BLEICHER,EDWIN
BLINN,JAMES C
BLUM,MRS MARION
BLY,JOSEPH A
BODEN,F J
BODNAR,J J
BOHACHEVSKY,I O
BONACHEA,R N
BOSWORTH,R H
BOWEN,EDWARD G
BOWEN,F W
BOWERS,J L
BOWYER,L RAY
BOYCE,W M
BRANDT,RICHARD B
BREITHAUPT,ALLAN R
BRITT,WARREN D
BROWN,COLIN W
BROWN,EARL F
BURROWS,T A
BUTLETT,D L
BUTZIEN,PAUL E
BYRNE,EDWARD R
BZOWY,D E
CABLE,GORDON G JR
CAMPBELL,J H
CANDY,JAMES C
CARDOZA,WAYNE M
CARRAN,J H
CASEY,JOSEPH P
CASPERS,MRS BARBARA E
CAVINESS,JOHN D
CHAFFEE,N F
CHAMBERS,J M
CHAMBERS,MRS B C
CHANG,HERBERT Y
CHANG,S-J
CHAPPELL,S G
CHEN,STEPHEN
CHERRY,MS L L
CHIANG,T C
CHIN,GEN M
CHODROW,MARK M
CHRIST,C W JR
CIRILLO,CARL
CLAYTON,D P
CLIFFORD,ROBERT M
CLOUTIER,J E
COBEN,ROBERT M
COCHRON,D E
COHEN,HARVEY
COLDREN,LARRY A
COLE,LOUIS M
COLLIER,ROBERT J
COLTON,JOHN R
COOK,THOMAS J
COPP,DAVID H
COSTANTINO,B B
COSTON,WALTER P
COULTER,J REGINALD
COURTNEY PRATT,J S
CRAGUN,D W
CRUME,LARRY L
CUNNINGHAM,STEPHEN J
DAVIDSON,CHARLES L
DAVIS,R L JR
DE JAGER,D S
DETRANO,MRS M K
DEUTSCH,DAVID N
DI MARSICO,BRIAN J
DICKMAN,B N
DICK,GEORGE W
DIMMICK,JAMES O
DIRKSEN,G E
DOMPIERRE,J A
DONOFRIO,L J
DOUGHTY,DAVID W
DOWD,PATRICK G
DREIZLER,HOWARD K
DRISCOLL,PATRICK J
DUBIS,M
EDELE,JAMES S
EDELSON,D
EDMUNDS,T W
EIGEN,D
EILENBERGER,ROBERT L
EITELBACH,DAVID L
ELLIOTT,R J
ELY,T C
ESSERMAN,ALAN R
FABISCH,MICHAEL P
FARGO,GEORGE A
FEDER,J
FELDMAN,STUART I
FELS,ALLEN M
FIGLIUZZI,MISS M E
FIORE,MRS RHODA J
FISCHER,H B
FLANAGAN,J L
FLANDRENA,R J
FLEISCHER,HERBERT I
FLUHR,ZACHARY C
FORMICA,JOHN F
FORTNEY,MRS VIRGINIA J
FORT,JAMES W
FOUGHT,B T
FOUNTOUKIDIS,A
FOWLER,BRUCE R
FOWLER,C F
FOX,R T
FOY,J C
FRANKS,RICHARD L
FRANK,H G
FRANK,MISS A J
FRANK,RUDOLPH J
FRASER,A G
FREEDMAN,M I
FREELING,M L
FREEMAN,R DON
FREIDENREICH,MRS B
FRITZSCHE,D
FROST,H BONNELL
FUCHS,EDWARD
FULTON,ALAN W
GALLI,A T
GARCIA,R F
GATES,G W
GAY,FRANCIS A
GEER,EUGENE W JR
GEPNER,JAMES R
GERARD,ALLAN
GEYLING,F T
GIBB,KENNETH R
GILBERT,MRS HINDA S
GIMPEL,JAMES F
GITHENS,JOHN A
GLUCK,F
GOETZ,FRANK M
GOLABEK,MISS R
GOLDSMITH,L D
GOLDSTEIN,A JAY
GORDON,P L
GOSNELL,MISS J
GRAHAM,R L
GRAMPP,F T
GRANDLE,J A JR
GREENBAUM,H J
GREENE,MRS DELTA A
GREENHALGH,H WAIN
GREISEN,MISS K E

+ NAMED BY AUTHOR

> CITED AS REFERENCE SOURCE

...
558 TOTAL

MERCURY DISTRIBUTION.....

COMPLETE MEMO TO:
10-EXD 13-DIR 135-DPH 127-DPH 8231-SUP 8234-SUP 9152-MTS 1352 1353 1356

COVER SHEET TO:
135 1271 1273 8234 5222
COOS = COMPUTING/OPERATING SYSTEMS/SURVEY PAPERS ONLY
COOSSI = COMPUTING/OPERATING SYSTEMS/ SYSTEM INTERCONNECTION, NETWORKS
UNOS = UNIX/OPERATING SYSTEM
UNSU = UNIX/SERVICE,UTILITY PROGRAMS

RADY,J E; MH 7B201;

TM-75-1352-2 TOTAL PAGES 21

TO GET A COMPLETE COPY:

. BE SURE YOUR CORRECT ADDRESS IS GIVEN ON THE OTHER SIDE.
. FOLD THIS SHEET IN HALF WITH THIS SIDE OUT AND STAPLE.
. CIRCLE THE ADDRESS AT RIGHT. USE NO ENVELOPE.

PLEASE SEND A COMPLETE COPY TO THE ADDRESS SHOWN ON THE OTHER SIDE
NO ENVELOPE WILL BE NEEDED IF YOU SIMPLY STAPLE THIS COVER SHEET TO THE COMPLETE COPY.
IF COPIES ARE NO LONGER AVAILABLE PLEASE FORWARD THIS REQUEST TO THE CORRESPONDENCE FILES.

Bell Laboratories

subject: Emulation of UNIX on Peripheral Processors

date: January 9, 1975

from: H. Lycklama
C. Christensen

TM-75-1352-2

Memorandum for File

Introduction

The Peripheral Processor concept allows a UNIX system (1) in a PDP 11/45 computer to be extended out to several peripherally attached PDP-11 processors. Each peripheral processor (PP) executes a regular UNIX process and has access to the central processor's (CP) file system and peripherals, yet does not contain a UNIX system. A process executing in a PP passes all UNIX system calls (read, write, create, etc.) to the CP for execution. This technique of partitioning a process at the UNIX system call level provides a clean, well-defined communication interface between the processors.

The hardware requirements for a PP are minimal compared to those for the CP. Since a PP doesn't require a resident UNIX system, it could have as little as 4K words of memory, depending on the size of the process to be executed. Since a PP executes only one process, segmentation hardware is not required.

Emulation of several PDP-11/45 instructions such as MUL, DIV, etc. allows the use of PDP-11/05's /10's and /20's as PP's.

Applications for a PP also cover a wide range. A mini PDP-11 might be an experiment or system controller, deriving its software support and data storage from the CP. A large PP could execute compute-bound programs, extending the processing capacity of the CP system. A variety of communication links between the PP and CP can be used. In the case of the large PP handling compute-bound programs the DEC LINK (2) device is appropriate. A PP controlling an experiment must be close to the experiment and could be connected to a remote UNIX CP via the SPIDER (3) network or, as in the system we describe, a serial I/O loop (4). A Data-phone connection could also be considered if a low data rate is satisfactory. A strong argument for supporting mini PDP-11's as UNIX peripheral processors is that a PP can be programmed in one of the high level languages available on UNIX, C (5), FORTRAN or LIL (6).

Configuration

The current configuration on which the PP concept is implemented includes three small PDP-11's attached to a UNIX CP through a serial I/O loop. The PP's are a PDP-11/10 with 8K words of memory used to control an experimental telephone system, a PDP-11/20 with 8K words of memory to be used as a front-end I/O processor, and a DEC GT-40 (PDP-11/05) to be used as a controller in an experimental digital filter system.

The CP is a DEC PDP-11/45 with 64K words of primary memory

and 96 megabytes of secondary storage. Other peripherals include three graphic terminals, six 113B data sets, one 201B data set with automatic calling unit, a connection to the SPIDER communication system, and a pair of Dectape drives. UNIX is supported on the CP by the MERT (7) operating system.

The serial I/O loop which connects all the PP's to the CP runs at an average rate of 3000 16-bit words per second. It is a message communication system, each message containing a 16-bit data word and a header specifying which PP the message is for. This I/O loop can support up to 63 PP's or other peripherals spaced at intervals of up to one thousand feet along the cable.

PP Communications Package

The PP communications package is a program which executes in the PP and handles all I/O loop communication with the UNIX CP. It is initially loaded from the CP by a bootstrapping procedure; execution of this package effectively connects the PP's local teletype to the UNIX CP as a terminal. This allows the PP user to login and run UNIX processes in the CP like any other CP terminal. A PP user sitting at the PP teletype can then edit, compile, assemble and run programs in the CP and ask the CP to load and run a program in the PP. Command dialog will be covered in detail later. The PP communications package does the loading and begins execution of a PP process under control of the CP. Then, when the process is running in the PP and a trap occurs, the communications package is called to alert the CP and handle CP requests for trap arguments and the return of trap results.

The PP communications package is dependent upon the type of PP to CP connection. In the case of the I/O loop it occupies about 400 octal bytes. A seven word PP bootstrap is used to load it from the CP, using the command "bootll -h", which must be issued from another CP terminal. "-h" specifies the PP to be booted. After the "bootll" command finishes and the PP communication package is started, the CP responds by typing the standard UNIX "login: " message on the PP's teletype.

Peripheral Processor Trap Handler

A version of the PP trap handler is prepended to each program that is to be executed in a PP. It catches all PP traps and passes those that it cannot handle to the CP via the communication package. This is the front-end package which must be link-edited with the object code produced by a UNIX compiler.

The trap handler includes code to determine the trap type (and, in the case of SYS traps, to determine the type of SYS trap). If the trap is an illegal instruction trap, the handler will determine if it has the capability to emulate this instruction, or whether it must be passed to the CP. If the trap is to be passed to the CP, a five word communication area in the PP is filled with the state of the PP at the time of the trap. The communication package causes an interrupt to occur in the CP, thereby alerting the CP process running on behalf of the PP. The PP trap state is then read from the communication area and upon processing this trap in the CP, the CP process passes argument(s) back in the communication area of the PP. Control is then

returned to the PP.

The trap handler also monitors the PP program counter and local teletype sixty times per second using the sixty hertz clock. This permits profiling of a program running in the PP and controlling it from the local teletype. Upon detecting either a rubout character (delete) or a control backslash character (quit) from the local teletype, a signal is passed back to the CP, causing the PP program to abort if these "signals" are not handled by the PP process. At the same time a check is made to see if there have been any delete or quit signals from the CP process. If the PP has no local teletype, setting a -1 in the switch register will turn control over to the CP process. If an undebugged program in the PP halts, restarting it at location 2 will force an IOT trap.

The trap handler consists of up to four separate components (see Appendix A for a detailed memory layout):

1. trap vectors, communication area, trap routines (400 words)
2. PDP-11/45 instruction emulation package (500 words)
3. floating point instruction emulation package (1000 words)
4. start up routine.

Of these, the first is always required. The illegal instruction emulation packages are loaded from a library only if required.

CP Emulation of Traps

During the time that the PP is executing a program, the associated CP process is roadblocked waiting for a trap signal

from the PP. Upon receiving one, the CP process reads the PP trap state from the communication area, decodes the trap and emulates it, returning results and/or errors. A check is also made to see if a "signal" (quit, delete, etc.) has been received. The CP process keeps a list of all of the current signals which are to be caught or ignored by the PP program, or caught by the CP process. If the PP is to catch a specific signal, control is then returned to the PP at the signal's entry point and the CP roadblocks waiting for another trap signal from the PP.

Of the more than 40 UNIX system (SYS) calls emulated, about 30 are handled by simply passing the appropriate arguments from the PP to the CP process and invoking the corresponding SYS call in the CP. The other 10 SYS calls require more elaborate treatment. Their emulation is discussed in more detail here.

The "getcsw" call (8) returns the CP's switch register, not the PP's, which is easily read without a SYS call. To emulate the "signal" SYS call, a table of signal registers is set aside in the CP process, one for each possible signal handled by UNIX. No SYS call is made by the CP process to handle this trap code. When a signal is received from the PP, this table is consulted to determine the appropriate action to take for the CP process. The PP program may itself catch the signals. If a signal is to cause a core dump, the entire PP memory is dumped into a CP "core" file with a header block suitable for the UNIX debugger.

The "stty" and "gtty" SYS calls are really not applicable to

the PP process, but if one is executed, it will be applied to the CP process' control channel. The "prof" SYS call is emulated by transferring the four arguments to the profile buffer in the PP memory. The PP, upon detecting non-zero entries here during each clock tick (60 times per second), will collect statistics on the PP program's program counter. Upon completion of the PP program, this data will be written out on the "mon.out" file. The "sbrk" SYS call causes the CP process to write out zeroes in the PP memory to expand the bss area available to the program. A SYS "exit" changes the communication mode between the PP and the CP back to the original terminal operation mode. It then causes the CP process to "exit" giving the reason for the termination of the PP program.

The three most time-consuming SYS calls to emulate are "read", "write" and "exec". The "exec" SYS call involves loading the executable file into the PP memory, zeroing out the bss area in the PP memory and setting up the arguments on the stack in the PP. A SYS "read" involves reading from the appropriate file and then transfering this data into the PP buffer. The SYS "write" is just the reverse procedure.

The "fork", "wait" and "pipe" SYS call emulations have not been written at this time and are trapped if executed in a PP. One possible means of emulating the "fork" call would be to copy an image of the parent process in one PP into another PP, permitting the "pipeing" of data between two PP's.

Forming a PP Program

The output of the C, LIL and Fortran compilers as well as of the assembler can be run on the PP's. The procedure is to compile the appropriate object modules using one or more of the following commands:

1. cc -c prog.c -> prog.o
2. lc -c prog.l -> prog.o
3. fc -c prog.f -> prog.o
4. as - prog.s -> a.out

and then link-edit in the appropriate trap handler, instruction emulators and start-up routines.

This link-editing is accomplished by means of the new program:

ldm [-mefp] [-f] prog.o [-lm]

Here [] indicates that the enclosed parameters are optional. The '-m' option (default) determines which one of the eight possible start-up routines is to be link-edited with the specified object modules to form the final 'a.out' file. The symbol '_main' must be defined in one of these modules as the program entry point. The various start-up routines include all possible combinations of the emulation package (e), floating-point package (f) and the profiler package (p). The default option '-m' will include none of these packages. The '-f' option specifies that the Fortran start-up routine is to be loaded along with the designated object modules. The symbol 'main' must be defined in the object

modules. In performing the 'ldm' command, libraries are searched in the following order:

1. user specified libraries
2. library of special PP run-time routines
   (in '/lib/libn.a')
3. C library routines
4. standard library routines.

If the '-f' option is specified the Fortran library is searched in place of the C library.

The mini run-time library "/lib/libn.a" includes routines to read and write the local teletype directly rather than by passing back the equivalent SYS "read" or SYS "write" to the CP. The illegal instruction emulation routines and three special start-up and clock routines are included in this library as well. The three routines are 'config()', 'sigtst()' and 'profile(&profbuf, pc)'. The configuration routine is used to specify the addresses of the hardware registers (control teletype, I/O loop, clock) which are dependent on the machine on which the program is to be run. This routine also starts the sixty Hertz clock. During the running of a program on a PP, the clock will interrupt sixty times per second and cause the execution of the 'profile()' routine, if profiling has been turned on, and of the 'sigtst()' routine. This latter routine checks for external signals from the local teletype, the I/O loop and a (-1) in the PP switch register. The user may provide his own versions of these

routines for different PP hardware configurations. In fact, he may wish to allocate the clock to some specific real-time function. This can be done by overwriting the clock interrupt vector with the address of the entry point of his clock routine.

Other special run-time routines exist in '/lib/libm.a' and may be loaded by means of the '-lm' option in the 'ldm' program. These routines include a special 'putchar' routine to direct the output from 'printf' statements directly to the local teletype. The "a.out" file generated by the "ldm" command can be run directly in the PP.

Loading and Running a Program on a PP

The command used to load and run a program on a PP is:

llr [-sij] prog [argl ... argn]

The first optional argument specifies on which PP the program 'prog' (typically 'a.out') is to be run. The default PP is the one from which the command was issued. If a configuration has a number of identical PP's, the user may run his program on whichever PP is available to him, i.e. he may invoke the scheduling option '-s'. However if the user's program can only be run on PP 'i' or PP 'j', he may specify the '-sij' option. A check is made to see if PP 'i' is available, and if not, the availability of PP 'j' is checked for. Locks are provided to avoid conflicting requests. Having determined on which PP the program is to be loaded, its text and data sections are loaded starting at address

Ø. The 'bss' section is appropriately zeroed out and the arguments (argl ... argn) are put on the PP's stack in UNIX fashion starting at the top core address available to the user program.

The program in the PP is started off at location Ø by the CP which then roadblocks waiting for a trap signal from the PP. In the PP, the initialization program 'config()' is executed before control is transferred to the user's program starting at the address '_main'.

The user sitting at his terminal has complete control over his program running in the PP. The "delete" or "quit" button on his terminal will abort execution or abort and produce a core dump of his program back in the CP. When running a program in a remote PP not attached to the user's terminal, the CP forwards any abort commands to the PP. If the PP has a local teletype, the abort commands can be issued from the PP local teletype or the user's terminal. If the PP has no local teletype, loading the control switch register with a (-1) will produce a core dump. For those undebugged programs which run wild or halt, restarting the PP at location 2 produces a core dump. A breakpoint trap may be planted deliberately in the PP program, producing a core dump upon execution.

For interactive control of a PP program, a symbolic debugger is available:

lld [-sij] prog [argl ... argn]

The PP program 'prog' is loaded in the specified PP and control is returned to the CP for further directions. Symbolic dumping, patching and planting of breakpoint traps may be done by the user. The flow of control of the PP program may be traced by planting multiple breakpoint traps.

Some Operational Statistics

Estimates have been made of the execution time of the various emulation routines. The times are approximate and assume a PDP-11/20 PP, a PDP-11/45 CP and an I/O loop connecting them.

The running times for the PDP-11/45 instructions emulated in the PP are as follows:

Inst.	usec.
mul	830
div	1200
ash	660
ashc	720
xor	440
sob	400
sxt	400

If execution time is important in a PP program, these instructions should be avoided. In C programs these instructions are generated not only when explicit multiplies, divides and multiple shifts are written, but also when referencing a structure in an array of structures. Using a PDP-11/35 or PDP-11/40 with a fixed point arithmetic unit as a PP would reduce the execution time for these instructions.

The average times to emulate floating point instructions in

the PP are as follows:

Inst.	usec.
add	2100
sub	2300
mul	3500
div	5600

For applications which require large quantities of CPU time running Fortran programs, it is possible to use a PDP-11/45 CPU with a floating point unit as a PP. More will be said about this in a later section.

For each SYS call the emulation package on the CP must read the communication area in the PP, emulate the actual SYS call and then return the arguments back to the communication area in the PP. Most SYS call's also require the passing up of some arguments from the PP. Typical times for a few SYS call's are listed below along with the ratio of time taken in the PP relative to the normal SYS call time in the CP:

SYS call	msec.(PP)	PP/CP
read	110	20
write	110	20
getuid	65	15
creat	160	5
open	150	5

Another data point is provided by a test program which copies 30,000 characters from one CP file to another. Running on the PP, the program takes 35 seconds, compared to 7 seconds on the CP. These times are strongly dependent on the data rate of the

communication link between the PP and CP. For the I/O loop used in the present configuration, the average data rate is 6000 bytes per second. It should be noted that most compute-bound and real-time PP programs do not require much CP communication. Indeed, communication is typically only required to access the file system on the CP.

Supporting Mini PDP-11s

Supporting a mini PDP-11 as a PP on a UNIX CP combines all the advantages of UNIX programming support with the real time response and economic advantage of a stand-alone PDP-11. Let's examine a typical PP programming session. A programmer sitting at the PP local teletype logs into the CP and uses the UNIX editor to update a PP program source file. It could be assembly language or one of the higher level languages available on UNIX (C, LIL, FORTRAN). Assume a C source file "prog.c". When the edit is complete the following commands are issued:

```
% cc -c prog.c
% ldm -me prog.o
% llr a.out
```

"cc -c" compiles the C program "prog.c" in the CP and produces the object file "prog.o". "ldm -me" combines the PP trap handler (-m) and instruction emulator (e) with the C object file "prog.o", generating an "a.out" object file. "llr" loads the "a.out" file into the PP, and starts it with the PP teletype as

the standard input and output. The programmer then observes the results of running the program or forces a core dump, and uses the UNIX debugger to examine it. If any program changes are required the preceeding steps are repeated. During this typical PP support sequence the programmer initiates the editing, compiling, loading, running, and debugging of a program on a mini PDP-11 without leaving its control teletype. It is the speed and convenience of this procedure along with the availability of high level languages which makes the Peripheral Processor concept a powerful mini PDP-11 support tool.

Some mini PP's will be disconnected from the CP when their software has been developed and the final product is a "stand-alone" system. Other mini PPs will always have a CP connection; they supply the real time response, unavailable from the CP, combined with access to the CP's software base, file system, peripherals, and connection to the computing community.

Some Possible PP configurations

Having shown the power of the emulation of UNIX on low-level PP's (e.g PDP-11/05 up to PDP-11/35), we will now consider the possibilities of emulating the UNIX operating system on a more powerful set of PP's (e.g. PDP-11/40 and PDP-11/45) with higher bandwidth communication channels to the central CP and making use of the scheduling algorithms developed previously. As a specific example, consider a PDP-11/45 CP with the full complement of memory, large secondary storage, and other peripherals. Now if this CP is compute-bound, it is feasible to connect one or more

PDP-11/45's as PP's. The appropriate communication link would be the DEC LINK device which permits the transfer of data on a cycle-stealing basis without intervention by the CPU. If these PP's each had 28K of memory (no segmentation unit necessary) and a Floating Point unit, they can be loaded with a Fortran-type or other compute-bound jobs, leaving the CP to handle the occassional file system requests. This multi-processor system would give good interactive response and have the ability to run compute-bound jobs without disturbing the rest of the system. It is also cost-effective, since the additional PP adds only about $35,000 to a total system cost of about $200,000. A 20% increase in cost yields a possible doubling in the throughput of the system. Yet the PP's have complete access to the CP file system. The addition of a few more PP's would increase the effective throughput of the system correspondingly. The PP's need not have local control terminals, rather they may be directly controlled by the CP. In running a configuration in this manner, there is no swapping overhead, since the PP program remains loaded until execution of the program is complete. This in itself can provide significant savings. From the PP user's point of view he has a dedicated processor working on his problem, with minimal load on the CP. Taking the compute-bound job off of the CP maintains the interactive nature of the CP.

Carrying the above example one step further, consider the running of a compute-bound job which requires more than 32K words of address space. A PDP-11/45 processor could be configured with

a segmentation unit and up to 64K words of memory as a PP on such a system. The PP could be loaded with a program which required the separation of I and D space, doubling the address space available to the PP program. This increases the cost of the PP, but it gives the user the ability to run processes which cannot be run in the CP.

The use of a PDP-11/45 with a segmentation unit and memory greater than 32K words opens up other possibilities as a PP. A SYS "fork" could be handled within a PP by copying an image of the process to another area of memory in the PP. All scheduling would still by done by the CP. There would exist a heirarchy of processes in the PP about which the CP would not need to know many details. For efficiency reasons, the PP trap handler could handle certain SYS call's such as "fork", "wait" and "break". However this requires parts of a UNIX operating system in the PP.

Emulation of Other Operating Systems

The concept of emulation of the UNIX operating system in a PP can be expanded to include the emulation of other operating system environments in a PP. This can be accomplished easily now by writing a new emulation package on the CP to handle traps from the PP. The trap handler in the PP has no knowledge as to what the operating system environment is, so that it can serve as a rather general-purpose trap handler for many different environments.

Summary

We have discussed the emulation of the UNIX operating system on a peripheral PDP-11 processor. This technique has facilitated the development of programs for the peripheral processors. In fact programs may be written in a wide choice of languages: assembler, LIL, C and Fortran. The PP's may range in complexity from a PDP-11/05 up to a PDP-11/45. The range of applications for the PP's covers real-time tasks, interactive tasks and compute-bound tasks. For this whole class of problems, programs may be developed on the CP and run on the PP under control of the CP and have access to the CP file system. Core dumps may be obtained to aid in debugging PP programs. UNIX software is available both during the development and running of PP programs.

We have discussed the implementation of this system using the I/O loop as the communication link between PP and CP. Other possible communication links include:

(1) data-phone

(2) DR11C's

(3) SPIDER system

(4) DEC LINK device (DR11B's back-to-back)

The LINK is now being programmed as a communication channel on the system described above. It offers a higher bandwidth communication channel than the I/O loop and should improve response significantly.

Acknowledgments

We are grateful to P.J. Plauger for his contributions to the peripheral processor concept and his aid in reviewing this document. B.C. Hoalst provided the PDP-11/45 instruction emulation package.

H. Lycklama

MH-1352-HL-JER

C. Christensen

Atts.
References
Appendix

REFERENCES

1. D.M. Ritchie, "The UNIX Time-Sharing System", MM 71-1273-4.

2. DEC PDP11 Peripherals Handbook.

3. A.G. Fraser, "SPIDER - A Data Communication Experiment", TM 74-1273-6.

4. D.R. Weller, "A High Speed I/O Loop Communication System for the DEC PDP-11 Computer", MM 73-1356-8.

5. D.M. Ritchie, "C Reference Manual", TM 74-1273-1.

6. P.J. Plauger, "LIL Reference Manual", TM 74-1352-8.

7. D.L. Bayer and H. Lycklama, "MERT - A Multi-Environment Real-Time Operating System for the PDP-11/45", memo in preparation.

8. K. Thompson and D.M. Ritchie, "UNIX Programmer's Manual - 5th Edition", June,1974.

APPENDIX A

PP Memory Layout

0-036	trap vectors
040-052	communication area
070-076	register save area
0100-0106	clock interrupt vectors
0110-0116	profile parameters
0120-0316	clock routine, user interrupt vectors
0320-0576	trap routines
0600-037376	start up routines
	PDP11/45 instruction emulation routine
	(optional)
	Floating point instruction emulation routine
	(optional)
	User programs
037400-037776	Communication Package