1122

Bell Laboratories **Cover Sheet for Technical Memorandum**

The information contained herein is for the use of employees of Bell Laboratories and is not for publication. (See GEI 13.9-3)

Title- **A Real-Time Time-shared Operating System for an SEL/86 - PDP-11 Configuration**

Date- **October 29, 1976**

TM- **76-8231-9**
76-1228-11

Other Keywords- UNIX
MERT

Author	Location	Extension	Charging Case- 38794
W. A. Burnette	**MH 2D-233**	**x4812**	Filing Case- 38794-23

ABSTRACT

A new operating system has been written for the SEL/86 - PDP-11 computer configuration in Department 1228. The primary purpose of the SEL - PDP-11 system is to provide real-time interactive computing for research in speech and graphics. The previous operating systems on these two computers supported a single real-time user in the same style as the DDP-224 computers in Dept. 1228. These operating systems have now been replaced by multi-user systems for more effective utilization of both the SEL and the PDP-11.

In this SEL/PDP-11 configuration, the SEL is connected via a high speed channel to the PDP-11 which controls several mass storage devices and provides the mass storage services for the SEL. The SEL controls a number of acoustic, graphical, and interactive devices.

For the PDP-11, the MERT operating system was adopted. System-level programs were written to provide communication between the SEL and the MERT file system. Simultaneously, MERT provides UNIX timesharing services. Thus users can access the same file system from either the PDP-11 or the SEL.

For the SEL, a new operating system was written with a foreground-background scheduler. The foreground supports the real-time user with uninterrupted processing until the program needs to wait for input from interactive devices. Background jobs are processed only while the foreground is waiting for user input. Background jobs may be prepared on remote terminals, using MERT/UNIX, and queued until the SEL is available for background processing.

In this way, the new SEL/PDP-11 system retains the single-user services for which this computer system was intended, but also provides a UNIX-style file system common to both machines, a UNIX time-shared program development facility, and a remote job execution and debugging capability for SEL programs.

Pages Text 12	Other 1	Total 13	
No. Figures 0	No. Tables 0	No. Refs. 7	

SEE REVERSE SIDE FOR DISTRIBUTION LIST

BELL TELEPHONE LABORATORIES, INC.

TM-76-1228-11
TM-76-8231-9

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

ARCHER,RUSSELL E,JR
BALDWIN,GEORGE L
>BAYER,D L
BELL,R E
<BLINN,J C
BROWN,W STANLEY
>CHRISTENSEN,C
<CRUME,L L
<CRUM,T A
DENES,P B
FLANAGAN,J L
FREEMAN,K G
<FROST,H BONNELL
FUJIMURA,O
GITHENS,J A
HALL,ANDREW D,JR
<HAMILTON,PATRICIA A
+HANNAY,N B
<HANSEN,GWEN J
HARR,JOHN A
HENIG,FRANCES H
JACKSON,H M,2ND
JULESZ,BELA
<KEESE,W M
<KERNIGHAN,BRIAN W
LIST,W H S
LONDON,H S
LUDERER,GOTTFRIED W R
>LYCKLAMA A NYEHOLT,H
MARANZANO,J F
MARCHETTO,CAESAR P
MATHEWS,M V
<MC GILL,R
MCILROY,M DOUGLAS
METZLER,HELEN M
MOLLENAUER,JAMES F
<MORGAN,SAMUEL P
MUSA,JOHN D
PINSON,ELLIOT N
POLLAK,H O
PRIM,R C
RIDGWAY,WILLIAM C,III
<ROCHKIND,M M
ROSENFELD,PETER E
<ROSLER,LAWRENCE
ROTHKOPF,E Z
SCHROEDER,MANFRED R
SINOWITZ,NORMAN R
SPIRES,R J
STERNBERG,SAUL

COMPLETE MEMORANDUM TO

STURIALE,P J
STURMAN,JOEL N
TAGUE,BERKLEY A
TERRY,M E
THAYER,P H,JR
TUKEY,JOHN W
<TURNER,VIRGINIA B
>WILLIAMSON,ROBERT W
WISH,M
59 NAMES

COVER SHEET ONLY TO

CORRESPONDENCE FILES

4 COPIES PLUS ONE
COPY FOR EACH FILING
CASE

ACKERMAN,A FRANK
AHO,ALFRED V
AHRENS,RAINER B
ALBERTS,BARBARA A
ALCALAY,D
ALLEN,JAMES R
ALLISON,C E,JR
AMITAY,N
AMOSS,JOHN J
ANDERSON,C R
ARNOLD,GEORGE W
ARNOLD,S L
ARTHURS,E
ARTIS,H P
ATAL,BISHNU S
BACCASH,JEANNE M
BACKUS,C F,SR
BARRESE,A L
BASEIL,RICHARD J
BATTAGLIA,FRANCES
BAUER,HELEN A
BAUGH,C R
BECKER,RICHARD A
BEL BRUNO,KATHLEEN A
BENNETT,JOHN EDWARD
BERGLAND,G D
BERNSTEIN,DANIELLE R
BERNSTEIN,L
BEYER,JEAN-DAVID
BIANCHI,M H
BICKFORD,NEIL B
BILOWOS,R M
BIRCHALL,R H
BIREN,IRMA B
BLAZIER,S D
+BLUE,JAMES L
BODEN,F J
BONANNI,L E
BOURNE,STEPHEN R
BOWEN,F W
BOWERS,JERRY L

COVER SHEET ONLY TO

BOWRA,JAMES W
BOWYER,L RAY
BOYCE,W M
BOYLE,GERALD C
BRADLEY,M HELEN
BRANDT,RICHARD B
BRANTLEY,JOAN T
BROWN,C W
BROWN,JAMES W
BURGESS,JOHN T,JR
BURNETTE,W A
BURROWS,THOMAS A
BYRNE,EDWARD R
CABLE,GORDON G,JR
<CALESSO,GIULIO L
CALLAHAN,R L
CANADAY,RUDD H
CANNON,T B
CARDOZA,WAYNE M
CARRAN,JOHN H
CARR,DAVID C
CASEY,J L
CASPERS,BARBARA E
CATO,H E
CAVINESS,JOHN D
CERMAK,I A
CHAFFEE,N F
CHAMBERS,B C
+CHAMBERS,J M
+CHANG,JIH JIE V
CHEE,T
CHEN,STEPHEN
CHEN,T L
CHERRY,LORINDA L
CHIANG,T C
CHODROW,M M
CHRIST,C W,JR
CLAYTON,D P
CLOUTIER,J
CLYMER,J C
COBEN,ROBERT M
COHEN,HARVEY
<COKER,C H
<COLE,LOUIS M
COLE,MARILYN O
COMMRADE,GERALDINE
CONNERS,RONALD R
COOK,T J
COPELAND,JOHN A,3RD
COPP,DAVID H
COREY,D A
COSTANTINO,BASIL B
COSTELLO,PETER E
COSTON,W P
CRAGUN,DONALD W
CRANE,R P
CSASZAR,MARYANN
CUNNINGHAM,S J,JR
DAVIDSON,CHARLES LEWIS
DAVIS,D R
DAVIS,R DREW
DE GRAAF,D A

COVER SHEET ONLY TO

DE JAGER,D S
DENISON,JOANNA
DERMOND,FAITH L
DEUTSCH,DAVID N
DEVLIN,N V
DICKMAN,BERNARD N
+DILLON,J F,JR
DIMMICK,JAMES O
DOBLMAIER,A L
DOLOTTA,T A
DONKIN,A E
DONOFRIO,LOUIS,J
DOWDEN,DOUGLAS C
DOWD,P G
DRENNING,C R
DRUMMOND,R E
DSTEFAN,D J
DUDICK,ANTHONY L
DUDLEY,ELIZABETH H
DUFFY,F P
DWYER,T J
EDELSON,DAVID
EDMUNDS,T W
EITELBACH,DAVID L
EL-SHABAZZ,A M
ELY,T C
EPLEY,ROBERT V
ERRICHIELLO,PHILIP M
ESSERMAN,ALAN R
ESTES,VIRGINIA DANIELLE
ESTOCK,RICHARD S
EVENSON,E K
FABISCH,M P
FACTOR,ROBERT M
FASCIANO,V
FAULKNER,ROGER A
FEDER,J
FEINBERG,HENRY R
FELDMAN,STUART I
FELS,ALLEN M
FELTON,WILLIAM A
FERRER,NANCY L
FETTE,C J
FISCHER,HERBERT B
FLANDRENA,R
FLEISCHER,H I
FOLEY,G
FONTENOT,SHARON M
FORTNEY,V J
FOUGHT,B T
FOUNTOUKIDIS,A
FOX,PHYLLIS A
FOY,J C
FRANKLIN,DANIEL L
FRANKS,RICHARD L
FRANK,AMALIE J
FRANK,H GREGORY
FRANK,RUDOLPH J
FRANZ,ANN M
FRASER,A G
FREEDMAN,MARVIN I
FREEMAN,R DON

COVER SHEET ONLY TO

<FREENY,ANNE E
+FRENCH,J C
FULTON,A W
GALLI,A T
GANNON,T F
GARCIA,R F
GATES,G W
GAWRON,L J
GAY,FRANCIS A
GELBER,CHERON L
GEORGEN,MICHAEL R
GEPNER,JAMES R
GEYLING,F T
GIBB,KENNETH R
GIBSON,H T,JR
GIBSON,T A
GILLON,ALEX C
GILL,R J
+GILMER,G H
GIMPEL,J F
GINSBERG,N
GLASSER,ALAN L
GLUCK,F G
GODWIN,R E
GOETZ,FRANK M
GOGUEN,N H
GOLABEK,RUTH T
GOLDSTEIN,A JAY
+GOODMAN,DAVID J
<GORMAN,J E
GORTON,D R
GRAHAM,R L
GRANT,GEORGE E
GRAVEMAN,R F
GREENBAUM,HOWARD J
GREENE,DELTA A
GREENWALGH,H WAIN
GRESHAM,J P
GROSS,ARTHUR G
GUIDI,PIER V
GUTHERY,S B
GUTT,DONALD J
HAFER,E H
HAGGERTY,JOSEPH P
HAHNER,MRS IRENE A
HAIGHT,R C
HALFIN,S
HALL,J T
HALL,MILTON S,JR
HALPIN,T
HAMILTON,LINDA L
HARKNESS,CAROL J
+HARRIS,JAMES R
HARTMANN,R H
HARTWELL,WALTER T
HARUTA,K
HAUSE,A DICKSON
<HAWKINS,DONALD T
HAYDEN,DONALD F,JR
HEDRICK,ELLEN L
HELD,R W
HELLER,J ALLEN

...
619 TOTAL

+ NAMED BY AUTHOR > CITED AS REFERENCE < REQUESTED BY READER (NAMES WITHOUT PREFIX WERE SELECTED USING THE AUTHOR'S SUBJECT OR ORGANIZATIONAL SPECIFICATION AS GIVEN BELOW)

MERCURY SPECIFICATION.....

COMPLETE MEMO TO:
8225-SUP 82-DPH 12-DIR 82-DIR 8231-SUP 8234-SUP 122-DPH 127-DPH

COVER SHEET TO:
1228-AMTS 1225-AMTS 8231-AMTS

COCSDC = COMPUTING SYSTEM DESIGN, CONSTRUCTION AND USE
UNOS* = UNIX/OPERATING SYSTEM

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.

BURNETTE,W A
MH2D233

TM-76-1228-11
TOTAL PAGES 13

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: **A Real-Time Time-shared Operating System for an SEL/86 - PDP-11 Configuration**
Case- 38794 -- File- 38794-23

date: **October 29, 1976**

from: **W. A. Burnette**

TM: **76-8231-9**
76-1228-11

MEMORANDUM FOR FILE

Introduction

Computer systems operated by department 1228 have typically been designed for single-user operation. Time sharing of these machines would detract from the real-time capabilities which are required for the speech input and playback devices, as well as from the responsiveness of these computers during tasks involving highly coupled interactions between graphical or speech displays and knob or sense-switch input devices. Such computer systems often become duplicated to make more machine time available, even though the individual components of each copy of the system are only used for a fraction of real time. For example, there are two copies of a system based on a DDP-224, each with two 7 megabyte disk drives, a tape drive, and several acoustic and graphical devices.

The SEL/86 computer has a fairly powerful processor and a memory with a 32-bit word length and 600 nanosecond cycle time. The SEL in department 1228 has a number of graphical and acoustic peripherals attached to it, but has no large mass storage devices of its own. Mass store services are provided by a DEC PDP-11 computer which is outfitted with disks, a tape drive, and a TIU interface to the Spider network[3].

With the SEL computer in department 1228, an effort has been made to depart from the traditional single user approach. The motivations are several: The SEL is more powerful (thus more likely to have excess capacity) and more expensive (thus more difficult to duplicate) than the DDP-224 systems. In addition, the unique architecture of the SEL/PDP-11 link suggests forms of timesharing which are more acceptible in the given real-time environment than more traditional timesharing systems would be. In particular, even though the SEL can not host both a real-time system for its console user and a full blown interactive system for remote users, a MERT/UNIX interactive system has now been provided on the PDP-11 while the SEL remains essentially dedicated to its primary user. SEL programmers can use UNIX commands to inspect and edit SEL programs at a savings of dedicated machine time on the SEL.

The SEL would still be underutilized if it served only its console user, because some of the console user's work involves user think time, and besides, the SEL is not operated 24 hours a day by a console user. A foreground/background scheduler for the SEL has been implemented to allow computer time unused by the console user to be used by non-interactive jobs sent to the SEL by remote users. Since this scheduler must give the ultimate priority to the SEL console user, it would not be useful to the remote users as a timesharing system. However, given that a timesharing system is available on the PDP-11, the background mode on the SEL is a satisfactory way to run compilations or non-interactive jobs, even long-running number crunchers.

Historical Background

The link between the SEL and the PDP-11 began simply as a means to provide mass storage facilities for the SEL which was delivered with no mass store devices. The PDP-11 was initially an 11/20 with 28K words of core, two disk controllers with a total of 5 drives of 2.4 megabytes each, a tape drive, and a network interface to the Fraser file store. In addition, the PDP-11 was equipped with low-speed communications interfaces and was capable of running an early version of UNIX.

Special intercomputer communication channels were built in department 1228 to connect the SEL with the PDP-11. These consisted of two data paths between the computers, one for control information, the other for file data. On the PDP-11, these were implemented as a DR11A single-word transfer device for the control channel and a DR11B direct memory transfer device for the data channel. On the SEL, both devices were implemented as direct memory transfer devices operated by the standard SEL channel hardware. One SEL channel is dedicated solely to the data channel device, while the control channel device shares a channel with other devices normally accessed by single-word transfers.

The PDP-11 was controlled by the SOPHI operating system[2] which was written to support the SEL by providing a logical path between the SEL and the data on the mass store devices. The file system (directory structure and space allocation), patterned after that on the DDP-224, was managed by the SEL, not the PDP-11.

The resident operating system in the SEL consisted of device drivers, a disk file manager, a command line interpreter, and a linking loader. The device drivers provided access to files on the PDP-11, files on the Fraser file store, the card reader, and the system console consisting of a keyboard and a Tektronics storage scope. Other devices on the SEL were driven by user programs which sent I/O commands directly to the computer's I/O channels. The command line interpreter parsed command lines read from a standard input file code, which could be attached to the console for interactive usage or to a file for canned jobs. The command interpreter loaded commands from either a list of resident commands or a library file containing non-resident commands. The loader, invoked as a resident command, accepted relocatable binary input from either sequential files or random libraries, leaving the loaded program in core, ready to run. This original SEL system was written by Barbara Caspers and was patterned after the DDP-224 operating system in use in department 1228.

The commands available on the SEL included a FORTRAN compiler, an assembler, a subroutine library editor, and a text editor. In addition, there were a number of utility programs for file maintenance, such as creating and releasing files, copying or reformatting files, etc.

No use was made of the hardware protection mechanisms available in the SEL. The user program was not prevented from scribbling in the system region or interfering with system I/O operations. This system certainly provided the maximum real-time response to a user program, but provided no error checking or system protection. Clearly, service for remote users could not be provided adequately without at least enough protection to keep the system from halting.

The New SEL/PDP-11 System

The mass storage support functions once provided by SOPHI on the PDP-11/20 are now provided by a program running under MERT on a PDP-11/45. Now users can access a common file system from either MERT or the SEL. MERT also supports a UNIX environment, thus providing a multi-user interactive system for preparing programs for the SEL.

The SEL system also provides for multi-user computing, but in a style which supports real-time service for a single console user. When the console user's program is not active, the SEL processes non-interactive programs for other users. As soon as the console user's program becomes active, the background user's program is swapped out on a fast disk, and the console

user's program is swapped in. The console user' program becomes inactive only while it waits for user responses on interactive devices. Interactive devices include the console keyboard and a set of knobs and switches. Special resident programs monitor these devices, allowing the foreground program to be swapped out until an appropriate signal arrives from one of these devices. As long as the foreground program is not waiting for an interactive device, it remains in execution, using the processor and data channels without interruption by background jobs.

Programs executing in the background have access to the same commands and subroutines as the foreground program. However, background programs are denied access to any of the peripheral devices on the SEL. The background mode can't be used for interactive computing. It may be used for compiling and debugging programs in preparation for foreground usage or for executing compute-bound non-interactive programs. A job spooling mechanism has been provided for remote users, using MERT/UNIX, to send jobs to the SEL background system. A post-mortem debugger for SEL programs (foreground or background) is available through UNIX.

The fast disk connected to the SEL is used for swapping, yielding a maximum exchange time of one third of a second when both programs are very large. The user switching mechanism has been designed to place the SEL console program in control as quickly as possible. Outstanding I/O requests generated by background programs do not prevent the foreground program from being brought into core because the PDP-11 provides buffering between the slow peripherals and the fast intercomputer channels. Outstanding I/O requests are simply held in the PDP-11 until the suspended background program is back in execution.

The operating system in the SEL invokes the protection mechanisms of the hardware. These protection mechanisms prevent user-mode programs from writing in the system area of core and from executing any instruction which could interfere with I/O operations. Thus the operating system prevents both the console user and the background programs from halting the system or interfering with the correct execution of other programs. The loader, command interpreter, and some other parts of the old system have been transported to the new system as user-level programs instead of system programs.

Using the MERT system, department 1228 now has the opportunity to switch some of the programs currently running on the DDP-224 to the MERT system, perhaps using subordinate processors. For example, LSI-11 microprocessors, or the PDP-11/20 replaced by the 11/45, may be connected to MERT to provide real time services for interactive programs running on the 11/45. For example, an Alles filter[4] has been connected to the PDP-11/20. With the 11/20 connected to the 11/45, programs on the 11/45, or even on the SEL, could access the Alles filter.

In summary, SEL programmers can use the UNIX timesharing services under MERT to prepare programs for the SEL, submit background jobs to the SEL, and inspect the results of running the SEL programs. The SEL console, available to only one user at a time, may be used to interact with SEL programs which require real-time response. These programs may, however, have been prepared and debugged (except for their interactive sections) using the SEL background mode. Programs for satellite PDP-11 processors may be written to service real-time devices for use by UNIX or SEL programs. All of these programming modes share a common file system maintained by MERT.

Pertinent Features of the SEL Hardware

Protection. The SEL is equipped with a memory protection feature and a protected/privileged mode distinction that can be used to protect the system memory and the I/O channels from the user program. The memory protection scheme provides a protection bit for each 512-word block of core, allowing the privileged mode to grant write permission to the unprivileged mode separately for each 512-word block. No relocation mechanism is provided, unfortunately. In the unprivileged mode, the computer cannot issue I/O commands, execute

the halt instruction, or write in protected memory. A set of trap interrupts detect attempted privilege violations and provide a system call mechanism. The mechanisms provided are sufficient to implement a protected system, although the loss of direct I/O capability in the user program may mean a degradation of real-time processing capability.

I/O channels. The department 1228 version of the SEL is equipped with four direct-memory access channels, each of which may be connected to several separate devices. Only one device may be active on a channel at one time, however. A channel is activated whenever a special I/O instruction is issued to it by the CPU. Prior to issuing the instruction, the CPU stores a buffer descriptor in a word in main memory dedicated to the particular channel. Upon receiving the I/O command, the channel fetches its buffer descriptor from memory and uses it to transfer data between the buffer in main memory and the device attached to the channel. When the buffer limit is reached or if the peripheral device terminates prematurely, the channel generates an interrupt at a memory location and CPU priority level dedicated to the channel.

A special feature of the SEL I/O channels, called "automatic reinitialization," permits buffer switching without interrupting the stream of I/O. When operated with automatic reinitialization (ARI), the channel will refresh its buffer descriptor from its dedicated memory location after each buffer is exhausted, without interrupting the stream of I/O to the peripheral. This feature can be used either to continuously retransmit a single buffer, or to transmit discontiguous buffers (e.g. alternating buffers) with no software overhead between buffers. ARI is used for output of continuous acoustic playback and for input of continuous data from a high speed optical scanner-digitizer.

Fast Disk. A pair of fixed head disks with a total capacity of 38 million bits, a maximum transfer rate of 270K words per second, and a maximum latency of 16.7 milliseconds have been attached to the SEL to be used as a swapping device and as a fast intermediate storage medium for picture processing. Used for swapping, the fixed head disk is capable of exchanging two programs, each using the entire user space, in a third of a second.

Miscellaneous. The SEL is equipped with a line frequency clock interrupt, and traps for references to non-existent memory, unimplemented instructions, memory parity errors, arithmetic overflows, and a large number of "external" interrupts that can be used as special signals from external devices or as programmed interrupt levels generated by a processor instruction.

Limitations of the PDP-11/20

Even though software existed for the PDP-11/20 for each of the two desired functions, timesharing and SEL support, there were clearly some stumbling blocks in implementing the desired system using the 11/20. First, usage of the PDP-11 would have to be restricted to well-tested system commands, since the 11/20 has no protection mechanisms. Second, UNIX and SOPHI were quite incompatible, and could not be merged without substantial recoding. SOPHI occupied 19K words, including buffers. Even if SOPHI's buffers were moved to extended core memory, the remaining code would not fit into the space left between the UNIX system and user command execution space. SOPHI would have to be entirely rewritten to be smaller and to interface with UNIX in the use of I/O devices. Third, the resulting system would have no room for growth, new devices, or new functions. Fourth, 11/20 UNIX was quite out of date and incapable of supporting more modern versions of commands. Finally, there still remained the problem of sharing files; the SEL and UNIX file systems were quite different, and neither was adequate for the needs of both systems.

An 11/45 processor would solve the memory size and system protection limitations and would support the latest UNIX software. For these reasons and partly for reasons beyond implementing the new SEL system, it was decided to replace the 11/20 processor with an 11/45. Using a modern timesharing system on an 11/45, some interactive tasks could be moved from the DDP-224 to the 11/45, leaving more time on the DDP-224s for tasks actually requiring a dedicated real-time machine. In fact, one could implement certain real-time

functions on processors subordinate to the 11/45, and the first such peripheral processor could be the 11/20 replaced by the 11/45.

UNIX vs. MERT for SEL Support

MERT was chosen over UNIX as the better operating system to provide SEL support. The primary advantages are the file system structure, which supports large contiguous I/O transfers, and the process modularity. Additional considerations were the real-time responsiveness available to MERT processes and the ability to connect subordinate processors[5] to the main machine. These capabilities could be used to provide some of the functions currently supported only by the single-user DDP-224 computers.

The directory structure of MERT is very much like that of UNIX, but file space is allocated in large chunks instead of single 512-byte blocks. The contiguous file structure coupled with the ability to do unbuffered I/O directly into user space allows MERT to retain the hardware's ability to perform long uninterrupted transfers, thus serving certain I/O bound tasks more efficiently than UNIX. Using the MERT file system, the SEL would have access to the UNIX-like directory structure provided by MERT and to contiguous files, as well, without managing its own file system.

The process structure of MERT permits dynamic loading of kernel- and supervisor-mode processes. In fact, most of MERT is implemented as a set of separately bound processes, many of which are dynamically loadable. Implementation of new system features, even device drivers, can be done without modifying the basic operating system, and tests and changes can even be performed without rebooting. MERT would therefore simplify the development of the new SEL support programs as well as new devices to be implemented in the future.

Architecture of SEL Support Under MERT

The SEL support system in the PDP-11 is naturally divided into-two functions: intercomputer communication and file system services. These two functions correspond to two types of processes under MERT: kernel processes and supervisor processes.

At least part of the SEL support program in the PDP-11 must run in kernel mode to operate the two intercomputer channels. Thus the intercomputer driver was implemented as a kernel process. A kernel process in MERT has access to the commands and registers used for I/O. The kernel process reads and writes intercomputer messages to the SEL and sends and receives these messages, via MERT interprocess communication facilities, to other parts of the SEL support system.

The remainder of the SEL support program was implemented as a MERT supervisor process. This supervisor process translates SEL requests into MERT file manager requests, including I/O requests, and sends the answers back to the SEL via the kernel process. I/O requests are buffered in the supervisor on their way between the peripherals and the intercomputer data channel. Within MERT, all processes obtain file system service by sending interprocess messages to the file manager. The SEL support supervisor obtains file manager services on behalf of the SEL by mapping intercomputer messages into their MERT interprocess message counterparts. Thus all of the file system functions available to MERT processes are made available to the SEL.

Rigid synchronization is needed between the control channel and data channel, and between the SEL and its support programs. This synchronization is achieved quite naturally using a slight extension of the usual MERT interprocess message. Each message sent or received across the intercomputer control channel is mapped into a MERT interprocess message transmitted between the kernel program and the supervisor program. For those messages which are used to synchronize data channel transmissions, the standard MERT I/O message is used, augmented to include the intercomputer synchronization message. MERT requires the

use of the I/O form of interprocess message to control the data channel I/O, as it does for any direct memory I/O transaction. However, the kernel process driving the intercomputer channels looks first at the intercomputer message embedded in the MERT interprocess message. The kernel process sends this data one word at a time through the control channel to the SEL. Then, if the message is a MERT I/O message, the kernel starts the associated data channel transfer.

The kernel device driver appears to the supervisor processes as a number of independent intercomputer channels to the SEL, one for each active user on the SEL. There are a number of copies of the SEL support supervisor running simultaneously, one for each user. MERT provides that a single copy of the instruction (text) segment of the supervisor program services all of the active copies of the supervisor. The SEL decides which copy of the SEL support supervisor is to be active. The SEL sends a message to the kernel program in the PDP-11 whenever it decides to switch users, and the kernel, in turn, wakes up the corresponding copy of the supervisor.

The Program Development Process Under MERT

Having decided on and ordered a new PDP-11/45, but having a 90-day delivery wait, one could still begin the program development process on the friendly neighborhood MERT system operated by the developers of MERT. On this system, the SEL support supervisor was coded and tested in action even without a live SEL to drive it. The modularity of MERT made it possible to have an ordinary user-mode program running under UNIX generate command messages to simulate the SEL and the device driver and thus to debug the supervisor.

The debugging facilities for MERT supervisors are roughly equivalent to those for regular UNIX programs. The undebugged supervisor rarely did anything bad to its environment. The two areas in MERT which were sensitive to supervisor misbehavior, file system permissions and supervisor process termination, have since been enhanced by the developers of MERT to render a wild supervisor process quite harmless.

Testing of the kernel process was not done until after installation of the department 1228 11/45. The most compelling reason for this is that the SEL connection hardware existed nowhere else. Another reason, however, is that there is little protection in MERT against damage by a wild kernel process, so it would be unfriendly to run an untested kernel process on a live system.

Elements of the SEL Programming Environment

At the outset of this project, it was decided to preserve the programming environment, that is, the command set and syntax and the subroutine libraries, of the old SEL system. This had been designed to parallel the DDP-224 and was considered important for purposes of moving programs and programmers from the DDP-224 to the SEL. Another compelling consideration was that it would take a lot of time to rewrite the compiler, assembler, loader, and other commands to adapt to a new environment. However, in the new SEL operating system, an attempt was made to implement the programming environment entirely in user-level code instead of designing the operating system to support only one type of environment. Thus some of the current SEL command structure and I/O methods have been transplanted from the old operating system to the new by moving them from the system area to the user area. In the new system, the old SEL environment has been preserved, but not cast in concrete. In the long run, one might consider implementing a UNIX-like environment in the SEL. Such an effort could be achieved almost entirely in user-level code.

The rest of this section discusses various transformations in the programming environment which were required to make it work in a multiple user protected system.

I/O structure. Some semblance of the old system I/O interface had to be provided in the new system. This interface was embodied in a half dozen or so subroutines and some associated system tables, "file control blocks." System-level and user-level code shared the information in these file control blocks, each level needing to look at and change certain fields. File control blocks were used for several functions: maintaining the disk file directory structure, remembering the position within a file, keeping track of packing buffers and pointers within buffers, flagging the type of file (ascii, binary, library, command library, etc.), and physical I/O parameters.

The original implementation of file control blocks would not be appropriate in the new system. For one, they permit the user program to change sensitive data. For another, the I/O structures supported by these file control blocks were too restrictive to be implemented at the system level. However, some notion of a user-level file table is necessary to support the programming structure chosen for user-level code. Files can be opened with various parameters by one command and used subsequently by another program which need not be programmed to know the file name or parameters. A typical programming style is to perform I/O by file number, letting the programmer choose the file just prior to program execution time by issuing an "open" command to attach that number to the desired file. Thus open files, with their various logical and physical I/O parameters, need to be preserved across commands.

The implementation chosen for the new system was to move the file control blocks to the user area and let the operating system ignore them. Each user has his own file control blocks, and they are swapped out along with the user program. A more general I/O interface to the operating system is provided, and the half dozen or so subroutines in the I/O library have been changed to map the file control block parameters into the appropriate system I/O requests. The current state of the I/O library is horrid, but at least future efforts to redesign it, should anyone wish to, will be easy because it all resides in user space.

Fortunately for the question of compatibility between UNIX files and SEL files, the format for text file data is identical in the two systems. There is a slight inconsistency; the SEL compiler and assembler need upper case input while UNIX is happier with lower case. To avoid a lot of file-to-file conversions, the SEL input subroutine was fixed to change text to upper case for the compiler, assembler, or any other command that needs it.

Command interpretation. The SEL command interpreter reads single input lines from a dedicated file number and parses the command name and the arguments. It builds an instruction sequence compatible with the SEL FORTRAN subroutine call and then searches for the program to execute the command. Certain commands are loaded along with the command interpreter and are called as subroutines. Others are looked up on disk, read in to memory, and then called as subroutines.

The old command loader kept its commands on a single command library file, and fetched them by looking them up in the library's directory. The new command interpreter uses the command name as a file name and attempts to load a command from the file.

The old command loader was a resident part of the operating system. The new one resides in user space, and thus can be swapped. Each user in the SEL has his own copy of the command loader.

Object files and program images. The linking loader on the SEL accepts relocatable binary input (in the form of special binary card images) from sequential or random libraries. A sequential library is just a seqence of binary decks as may be produced by a series of compilations or assemblies. A random library is a collection of binary decks preceded by a directory containing all external names satisfied by programs in the library along with pointers to the particular binary deck.

The loader in the old SEL system left the resulting program (and load map) in core, ready to be transferred to via the $RUN command. The loader was modified in the new system to store the program and map on a file in command-loadable format. System commands are installed simply by moving this file to the appropriate directory where the command loader will find it. The load map is stored on the file for successive steps in loading the program and also for debugging purposes.

Program errors and termination. The old system provided no system protection, and the majority of programming errors would result in halting the machine, sometimes after wiping out the system. The new system provides memory protection and also enables trapping of the hardware-detectable program errors. In addition, the new system provides a program timer which may be used to abort the program or signal it when the timer runs out. Program aborts cause a core image dump to be written (in the same format as a command) and an error status to be returned to the command loader which then prints the termination message.

Console keyboard/display. The console, consisting of an ascii keyboard and a Tektronics storage scope, was programmed in the new system to be alive at all times (full duplex). Input characters are echoed on the scope, and then stored in a queue until a program reads them. Certain characters have special meaning to the system. One special character erases the scope, two other characters provide the line and character delete functions. One key provides an "interrupt" which may be caught or ignored by the program (or else the program is terminated), and another provides a "quit" which unconditionally terminates the program with a core dump. One important function provided by the quit and interrupt keys was available previously only by manually halting the SEL: stopping a looping program.

Direct I/O. Programs developed on the old SEL system for the various special acoustic and graphical devices relied on the ability to issue I/O commands directly to the channels. In a protected environment it is necessary for the operating system to control I/O, both to prevent interference with other I/O activities and to prevent I/O transfers into system memory. The protected mode of the SEL processor prevents direct execution of I/O commands. A system interface was provided to let user-programs specify I/O commands to be issued by the system to the special devices after making sanity checks.

Successive Stages of the New SEL System

The new SEL system was not installed all at once. It was updated in three major steps from the original SOPHI link to the current foreground/background system.

The first step was to get the SEL running concurrently with timesharing. In this first step, the smallest possible number of changes were made to the SEL operating system to get it running with the new SEL support system. In this mode, the SEL still maintained its own file system, using only the MERT files /dev/rk2, rk3, and rk4, the three disk drives that had been dedicated to SOPHI. It was this system that was first used to test the SEL device driver. With a few more minor changes, it became possible for the SEL to communicate with ordinary MERT files, if their names could be expressed in eight or fewer characters (to fit in with the existing command syntax and file control block format). At this point, simultaneous operation of MERT and the SEL and limited file sharing between the two environments became possible. This halfway point in the project immediately improved the SEL program development process.

The next step was the first attempt at rewriting the SEL operating system. The major goals for this system were:

1. That it invoke the hardware memory protect mechanism.
2. That it allow access to the full range of the MERT file manager functions (i.e. creating files, arbitrary length names, changing current directory, etc.).

3. That it serve as the foundation for the final step: the foreground/background system.

Goals 1 and 3 required the reorganization of system versus user areas. Goal 1 also created an additional problem; programs which depended on running in privileged mode to do I/O would no longer run in the protected environment. A general interface to the I/O channels had to be provided. The alternative of resident device drivers for each device was rejected because, for one reason, too much core would be used for functions rarely invoked. For another, development of new device drivers would be easier done in user space if a general I/O interface between the channels and user-level code were available. Goal 2 required some work in the area of command syntax and user I/O subroutines as well as completely redesigned system-level programs which interface to the SEL support system in the PDP-11.

The final step was the introduction of swapping, user switching, and scheduling to implement a foreground-background system. The scheduling scheme chosen for this system is quite simple. Only one job occupies core at once. If a higher priority job becomes active, the current job is swapped out and the new job swapped in. There are only three jobs in the system, and their priorities are fixed: primary user highest, then one background region for short jobs (as measured and enforced by the system clock), then another background region for long jobs. Jobs for background are spooled from a queue of job files in a directory dedicated to job queueing.

One main goal in the design of the user-switching mechanism was that it allow very quick preemption of a background user by the primary user. In fact, the SEL need not wait for all outstanding I/O transactions into the background user's program to complete; the real peripheral data transfer is buffered in the PDP-11, and only the transmission currently in progress on the data channel between the PDP-11 and the SEL need be waited for. The entire user switching process takes a maximum of one third of a second, depending on how large are the two programs being exchanged.

The highest priority program which is "active" is the one which is given control. A program remains "active" until it needs to wait for some external event which is expected after a waiting period longer than the time it takes to swap. The typical background job has no such events to wait for. The primary user's program, however, must wait for responses from the interactive devices, such as the keyboard/scope console. The primary user program will become temporarily inactive while waiting for a resonse from the user on the keyboard or while sending a large number of characters to the scope display. While the primary user's program is inactive, the highest priority active background job may be swapped in.

A certain amount of effort has been spent on determining when a program should become active or inactive. The more the primary program is active, the less time is available for background jobs. Yet, response time for the primary user program is considered very important for certain interactive functions. For example, when a display is being controlled by interactive devices, such as knobs or switches, the program should respond very quickly to the interactive devices. However, if the console user is staring at a display for a long time without making changes, the background program should be able to get control. Currently, a resident monitor program in the SEL can be invoked to watch some subset of the interactive knobs and switches and wake up the primary program when some threshold of change takes place. This scheme seems to work well, given the fast user switching mechanism.

Architecture of the New SEL System.

There are five rough categories of processing in the new SEL operating system:

1. I/O channel control, interrupts, and traps.

2. Device drivers (fixed head disk, control and data channels, console keyboard and scope) and the general I/O interface for devices without system-level drivers.
3. Interface to the support program in the PDP-11.
4. Event detection, process scheduling, and command startup and termination.
5. User interface via system calls.

The following discussion is intended only to mention highlights of the system, not to fully document its workings. Further details may be from a separate memo on the subject of implementation of the new system.

I/O channels and traps. Since the devices on the SEL share common channels, I/O requests from the various device drivers must be scheduled centrally. The device drivers send requests to the central I/O system in eight-word blocks which are queued for the appropriate channel. The queued blocks contain the channel commands to be issued as soon as the channel is available.

Since interrupt processing can itself be interrupted by higher level interrupts, and since no hardware stack for the processor exists, interrupt processing must be fairly carefully scheduled. "Critical regions" in the operating system are protected against interruption by activating an otherwise unused very high priority level. All interrupts and traps are received by the central I/O system. The interrupts are then passed to the device drivers via the same queue blocks originally sent to the I/O system.

Every activity on an I/O channel is timed. Certain devices may behave strangely if not operated properly, so a channel timer protects the system from the possibility of losing an interrupt and having the channel forever unavailable. When a 10 second channel timer runs out, the I/O request is terminated with an error status, and the channel is made available for the next I/O request.

Device drivers. Device drivers are provided for the devices which are frequently used. There is also a device driver for a class of devices which are not frequently used, but are used for operator input and must be constantly polled if a program is using them. Examples are the four knobs and six sense switches. The resident device driver does the polling so the background program can swap in while the primary user is thinking about which knob to twirl next.

To access devices not handled by resident device drivers, a general interface to the I/O channels is available at the user level to generate direct commands to the channels. First, the commands are checked for validity and the data transfer range is checked to enforce the user program address limits. Then the command is passed to the I/O system to be scheduled for the appropriate channel. At this point, the user program may resume execution or wait for the I/O to complete. When the interrupt occurs, the device status is obtained and passed back to the user program.

Interface to the PDP-11. This interface consists of device drivers for the control channel and data channel plus a queue handler for file system activity. Requests to the file system are reformatted here from simple argument lists into the appropriate intercomputer command sequence. Certain special intercomputer commands for user switching and for sending unexpected events (i.e. events not due to file system requests) between the two machines are also processed here.

Process scheduling. Some of the functions included here are loading and termination of processes (commands) and producing the core dump, handling timeouts, program-generated traps, signals, and preemption by higher priority jobs.

There are only three jobs known to the system at a time: the foreground job and two background jobs. The priority relationship among these three jobs never changes. The job in the foreground has highest priority, then the job in the background queue for short jobs. The only decision made by the scheduler is to determine which jobs are actually active. The highest

priority active job is then brought into execution.

The first process after system initialization for any of the three priority levels is an initialization process which decides the source of command lines and the destination of messages and then invokes the login and command interpreter processes. For the foreground user, the console is initially the source of commands and the destination of messages. For the background jobs, the initialization process periodically examines the background job queue to find the command line source. When the command interpreter terminates, the system invokes a new copy of the initialization process.

Processes can invoke other processes in a push-down fashion, but the core image of the calling process is not saved during this procedure (e.g. by swapping it out*). The loader and command interpreter work by being loaded lower in the user region than the processes they call.

System calls. The CALM instruction provides the trap to the system call processor. A variety of system calls are provided. These include file system calls, I/O, setting signals and timeouts, loading and dumping core images, and reading certain system-maintained values (e.g. the clock, the current user number, etc).

Growth and New Facilities

A number of changes in the hardware configuration of the PDP-11 were made because of the new timesharing capabilities. A floating point option was installed, and an additional 32K of memory is on order, which will bring the total memory of the 11/45 to 104K words. New disks were installed to increase online file storage space and improve response time.

The new disk controller was purchased from Xebec Systems, Inc. (model 7600) with two drives. Each disk (CDC 9762) holds 60 megabytes of data, twenty five times the size of an RK disk. These drives have a rotational rate of 3600 rpm and an average seek time of 30 milliseconds. A MERT device driver was written for the Xebec disks and these replaced the smaller disks on the MERT file system. One RK controller with three drives was subsequently moved to the 11/20 processor. The other RK controller remains on the 11/45 for backup, diagnostics, and booting.

The users have enriched the SEL programming environment by installing a RATFOR[6] preprocessor, an SEL postmortem debugger[7], and facilities for submitting jobs to the SEL. The command set of the SEL is not very well developed, but UNIX provides the services lacking in the SEL. A direct connection between the SEL console and UNIX would conveniently eliminate the need for any sophisticated commands on the SEL. Currently, the same end is accomplished by having an ordinary terminal sitting by the SEL console.

A generalized notion of interprocess pipes, perhaps managed by the MERT file manager, would make communication between the SEL and UNIX much more powerful. For one, pipes could be used to transmit commands and responses between the SEL console and UNIX. More generally, it would permit cooperation between ordinary processes running concurrently in the SEL and the PDP-11.

Conclusion

The interactive and foreground/background systems on the PDP-11 and SEL/86 provide a unique mixture of programming services to the SEL programmer. Both time-shared and real-time services have been made available by providing them separately on different computers. SEL programmers now have the benefits of UNIX and the power of the SEL processor, side by side, sharing a common file system.

---

* This could be done at the cost of more space on the fixed-head disk. Currently, one tenth of the total disk space is dedicated to the swap areas for three maximum-sized programs.

The intercomputer connection was implemented in quite a natural way within the architecture of MERT. The concept of separate processes, including a file manager, using interprocess messages to communicate between them, was used to solve the problems of intercomputer communication.

The new SEL operating system sacrifices little of the real-time responsiveness of a dedicated machine, yet allows more machine utilization by supporting multiple users. The SEL programming environment has been enhanced by the provision of UNIX services and will continue to blossom under the influence of the UNIX programming environment.

W. A. Burnette

W. A. Burnette

MH-$\frac{8231}{1228}$-WAB-troff

Att.
Ref

References

[1] D. L. Bayer and H. Lycklama, *MERT - A Multi-Environment Real-Time Operating System*, TM-75-1352-7.

[2] R. W. Williamson, *SEL-PDP Operating Configuration: SOPHI - The PDP Operating System*, TM-73-1228-3.

[3] A. G. Fraser and J. R. Vollaro, *Specification for Connection to the Spider System*, TM-72-1273-2.

[4] H. G. Alles, *A Multi-User General Purpose Laboratory Digital Filter*, TM-75-1356-4.

[5] H. Lycklama and C. Christensen, *Emulation of UNIX on Peripheral Processors*, TM-75-1352-2.

[6] B. W. Kernighan, *RATFOR - A Rational FORTRAN*, TM-74-1273-16.

[7] J. R. Remde, *Two Utility Programs for Submitting and Debugging Remote Jobs for the SEL*. TM-76-1228-8.