CMU-CS-71-104

A SOFTWARE LABORATORY

PRELIMINARY REPORT

K. Corbin E. Hyde
W. Corwin K. Kramer
R. Goodman E. Werme
W. Wulf

August 23, 1971

Carnegie-Melion University

Pittsburgh, Pennsylvania

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0107) and is monitored by the Air Force Office of Scientific Research.

ABSTRACT

This report describes the implementation of the kernel of a simple multi-process operating system. The purpose of this system is to create an environment for the construction of experimental programming systems for educational and research uses.

INTRODUCTION

This report describes the initial design of a "software laboratory". The objective of this system is to create an environment within which researchers and students may experiment with the construction of software systems. The system accomplishes this, providing a large number of functional "modules" together with a mechanism for flexibly interconnecting them in various ways. The philosophy of the system is a software analog of the hardware "macro-modules" of Clark [1] and "register-transfer-modules" of Bell [2]. Much of the philosophy for the approach described below is due to Krutar [3]; key ideas were borrowed from Habermann and Jones [4] and from many discussions with Per Brinch Hansen.

The similarity between many of the components of various systems programs has often been noted, but seldom exploited. Lexical analyzers and syntax analyzers, for example, occur in all compilers, and to some extent in assemblers, editors, command interpreters, etc. Yet they are generally re-written for each such system (translator-writing-systems, or compiler-compilers, have been the one exception to this practice. This situation is especially annoying to two groups of people to whom the present report is primarily aimed: (1) the researcher who would like to quickly fabricate a system in order that he might pursue a single aspect of it in depth, and (2) the instructor who would like to assign programming problems on some aspect of systems programming but which only make sense in the context of a complete system. To illustrate the point, consider the researcher (or student) who would like to (is assigned to) investigate various compiler optimization strategies on the tree-representation of a program. To do this lexical analysis, symbol table, space management, parser, tree-generation, and i/o functions must first be written. None of these is essential to the project at hand, and collectively they may be sufficiently effort-consuming to make the project impractical.

One purpose of the project is to provide an inventory of functional modules such as those mentioned above -- several lexical analyzers, parsers, etc. -- and an environment in which they may be quickly interconnected. Thus the researcher (or student) may quickly compose a host environment for the particular sub-system of interest.

The system has been implemented on a minimal PDP-11 configuration in order to make it widely available. Future reports will specify modules and exercises suitable for intermediate and advanced software laboratory courses. This preliminary report deals exclusively with the environment -- its philosophy and the construction of its "kernel."

THE PHILOSOPHY

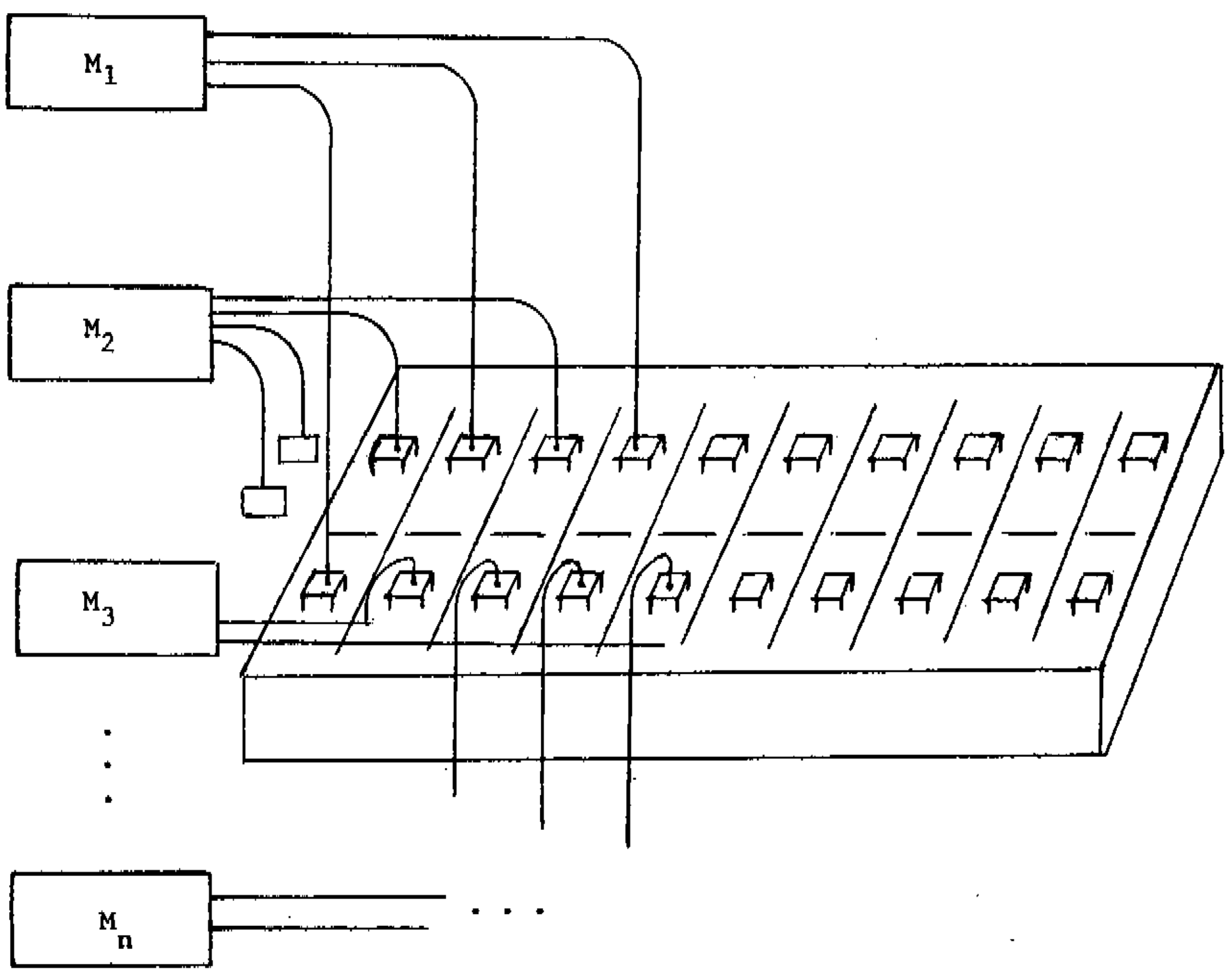
The philosophy of the environment created by the system comprises the consequences of a particular physical model which we would like the user to have of that environment. That model is:

> A (user) system is constructed from a number of components called modules. A module is a functional unit which receives signals (data) along one of a number of input wires, cables, or ports, performs some operations and (possibly) generates output signals on other cables (or ports). The cables connected to a module are fitted with standard male/female connectors so that the output of any module may be directed to the input of any other by appropriate interconnection of their cables. Rather than direct interconnection, a special "patch panel" similar to an old-fashioned telephone switchboard, is provided to facilitate the interconnections. Figure I illustrates this model.

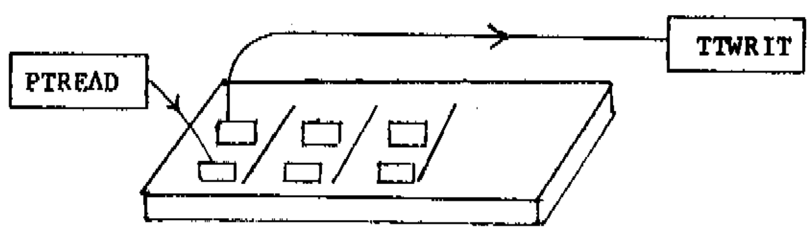
In this model modules do not know to whom or what they are connected. They use internal names to reference ports for receiving and sending information and the actual supplier or receiver is specified externally by the particular cabling pattern established by the user. This fact coupled with the "standard connector" assumption permits the substitution of a module for a functionally equivalent one (or network of ones) at any time.

Figure I

The Physical Model



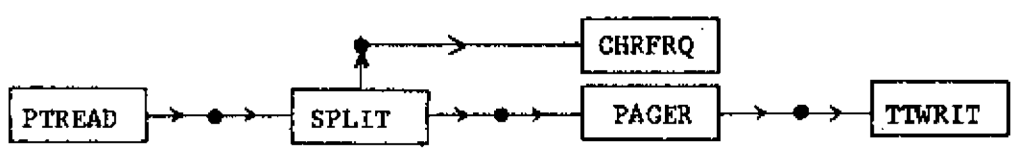
The use of the system is best illustrated by a simple example. Suppose one wished to construct a program to read text from a paper-tape reader and print it on the teletype. Modules exist for reading (characters) from the paper tape reader (PTREAD) and writing (characters) on the teletype (TTWRIT) -- so they can be interconnected as follows:



Suppressing the patch panel helps to clarify the diagram in more complex examples, so let's draw this configuration as simply

Now suppose we would like to add pagination of the output. Further, suppose we have a module (PAGER) which accepts input and passes it along to its output, but also looks at each data item for a special end-of-line (EOL) character, counts them, and after the nth inserts several special upspace-the-paper (line-feed) characters. If we break the original connections and reconnect as shown below we will now get the desired pagination.

Suppose further, now, that we would also like to get a character frequency distribution in the text while the printing is going on. If we happen to have a module (CHRFRQ) to do this we might create the following configuration:



In this configuration 'SPLIT' is a simple module which, when it receives input, replicates that same input on each of two output ports. We could proceed in this way to build much more complicated configurations but trust that the example has served to illustrate the general philosophy.

Of course, software modules are not physical objects; they do not have tangible cables dangling out of them. The patchboard does not have a physical existence either. Thus, the acts of connection and reconnection are not accomplished by physical acts, but rather by commands typed on a terminal. The precise syntax of these commands is beyond the intended scope of this report, and in any case is likely to change as more attention is paid to the human engineering aspects of the system (which we consider to be a crucial aspect of the whole project). Suffice it to say that the structure of these commands is intended to reinforce the conceptual model presented above. Thus, the commands mimic the things one would expect to do to modules physically wired together -- for example connections may be made or broken at any time, the complete "wiring list" may be displayed or individual wires traced, the signals flowing along a particular cable may be monitored, etc.

THE IMPLEMENTATION APPROACH

The system model presented in the previous section might be implemented in any one of a number of ways each module could have a subroutine or co-routine structure, for example. Rather than either of these it was decided to construct each module as an asynchronous sequential process. The cabling and patchboard are implemented as a "mailbox" message buffering system. The system is implemented in two pieces: (1) a small "kernel" which includes space management, process management, and message handling primitives, and (2) a "user representative" which implements the command language, tracing, loading of modules, displays, etc. The user representative (UR) is implemented as a set of modules using the mechanisms provided by the kernel. It is in no way different from, or more privileged than modules assembled by the user. This construction philosophy permits the UR to be easily modified, permits different versions of the UR for different users, and permits the UR to be easily adapted to various configurations and needs. A continuing aspect of this research is the human engineering of the UR — built as a set of modules, it permits this type of experimentation to be done in its own environment. Finally, the UR, being constructed from modules itself, forms an advanced example of the use of the system.

The kernel has been purposely kept small and "clean" (the entire kernel consists of less than 200 PDP-11 instructions. The small size of the kernel allows (1) the design and implementation to be iterated, and

(2) the kernel itself to be an object of study in a systems programming course, and (3) a usable subset of the total system to be used on a minimal (4K) PDP-11 configuration.

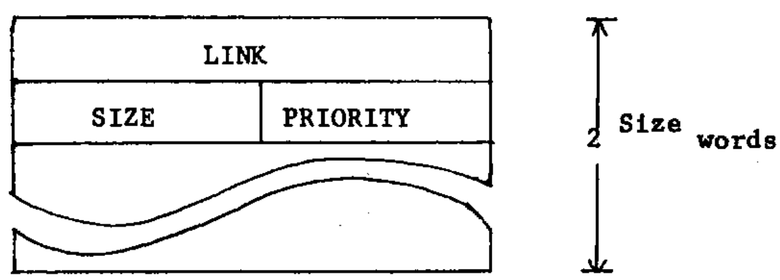
IMPLEMENTATION OF THE KERNEL

The kernel consists of a small number of data structures, accessors, and routines for manipulating the structures. The data structures used in the kernel are instances of a smaller number of "classes" of structures (objects, lists of objects, semaphores, and vectors). The routines in the kernel are constructed such that each performs an operation appropriate to a class of structures on any instances of a member of that class; that operation is never performed by any other routine. The immediately preceding sentence may be interpreted as a working definition of the term "clean" used earlier. It should be noted that this use of "clean" conflicts with that proposed elsewhere [7] in that it implies a strong functional interdependency, and some loss in efficiency; it was chosen in favor of a (data) semantic interdependency because of the clarity and modifiability it affords.

The following description of the kernel is divided into an English description of the data structures and their associated manipulative routines, and a Bliss module which implements them. The latter is to be considered the authoratative definition of the kernel.

(1) Objects

An "object" is a data structure which is composed of 2^n $(1 \le n \le 16)$ words, two of which contain a link field (objects are frequently chained together on lists), size field (contains n when actual size in 2^n), and priority field (when on a list, objects are always in priority order).



The routines for manipulating objects are:

a) get (n) — allocate memory for an object of size 2^n and return its address

b) release (a) — deallocate the space for an object whose address is 'a'. The value of 'release' is undefined.

c) copy (a,b) — copy the contents of an object whose base address is 'a' into an object whose base address is 'b'; at most, size (b) words will be copied. Return the base address of 'b'.

d) newcopy (a) — create an object and make its size and contents identical to those of 'a'; return the address of the new copy.

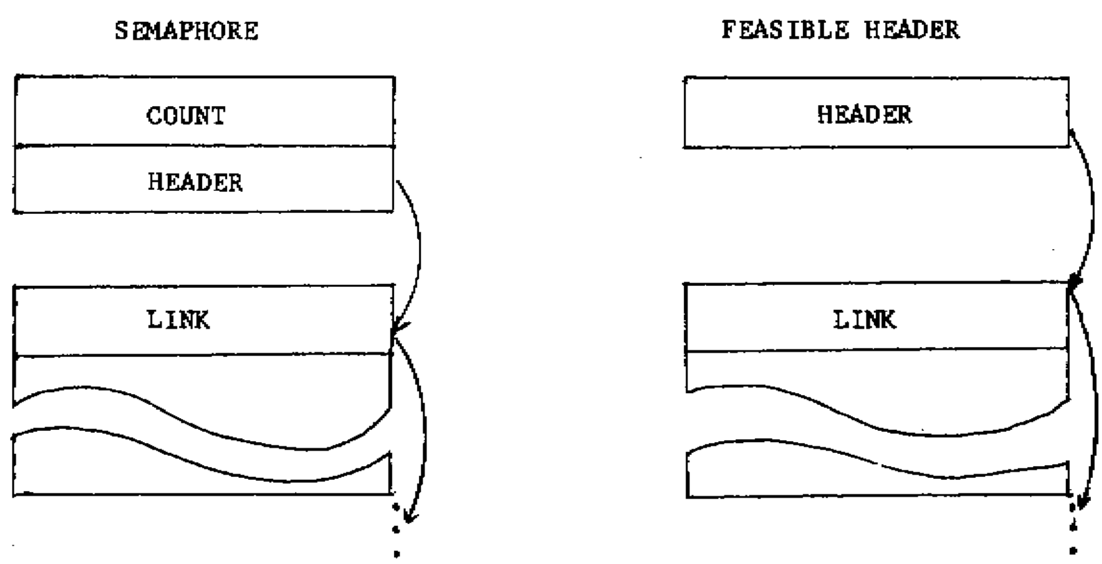
e) link (a,h) — link the object whose base address is 'a' onto the list whose header address is 'h'. The object will be linked into the proper priority position on the list. Return the address of 'a'.

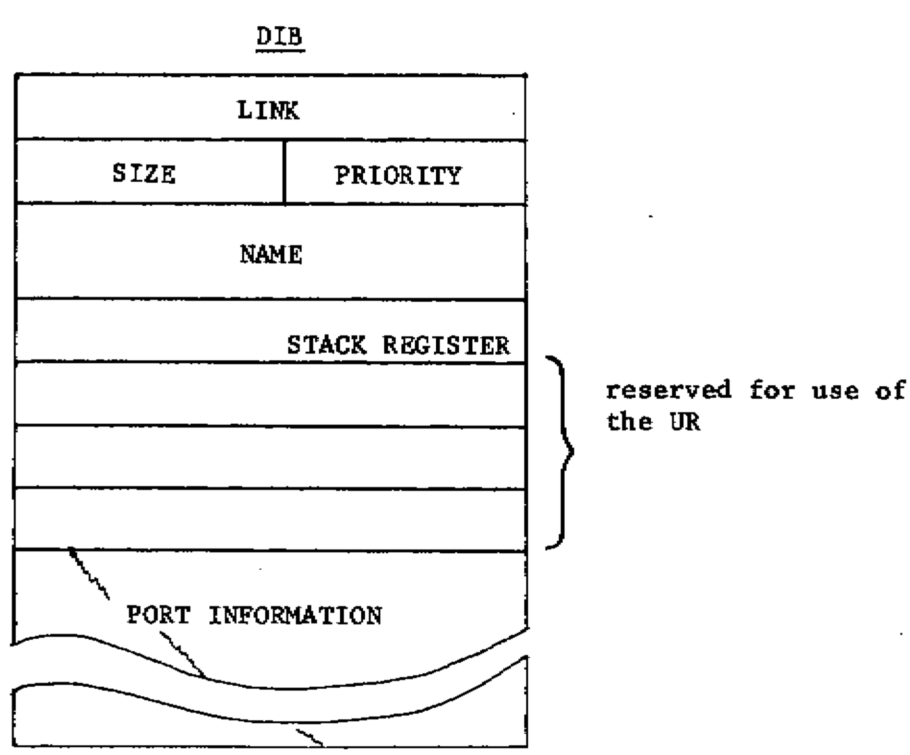
f) delink (h) — remove the first object, that is the highest priority one, from the list whose header address is 'h' and return the address of this object.

g) swap (h1,h2) — delink the first object of the 'h1' chain and link it onto the 'h2' chain; return the address of the swapped object.

(2) <u>The 'feasible' list, semaphores, and synchronization</u>

A particular class of objects are called "DIB's", <u>d</u>ynamic <u>i</u>nformation <u>b</u>locks. A DIB is the name given to what has been called a 'process description' in other systems, and contains relevant state information for a process. The 'feasible' list is a chain of all the DIB's for processes which are ready to run. All other processes are "pending on a semaphore" and these DIB's are chained on a list associated with that particular semaphore. The reader is assumed to be familiar with Dijkstra's P and V primitives and their use for process synchronization [6].





The routines which manipulate semaphores and the feasible list are:

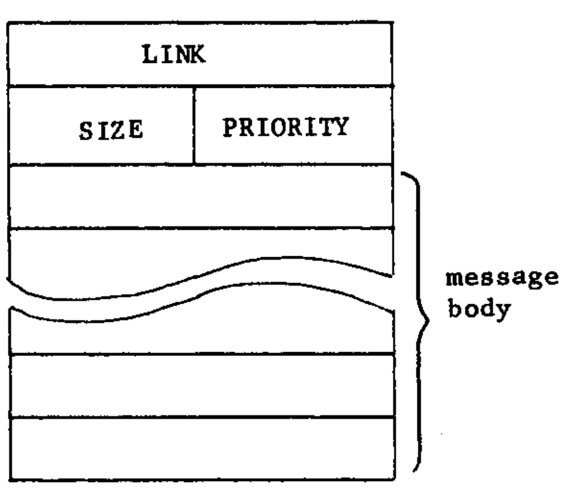
savstart	saves the context of the current process on its stack, saves the stack pointer of the current process in its DIB, and initiates the process whose DIB is at the top of the "feasible" list by first retrieving its stack pointer and then restoring its context
P (sem) V (sem)	Dijkstra's synchronization primitives

(3) Messages, Mailboxes, Ports, and Communication

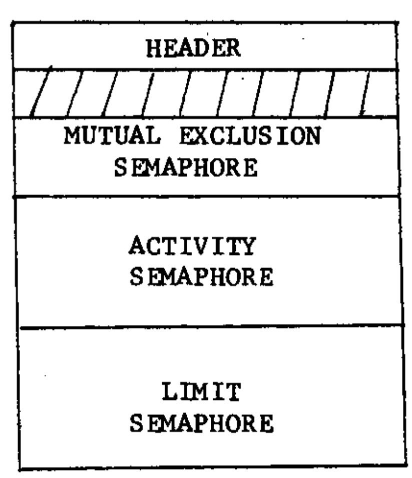
Processes communicate by sending and receiving objects called "message " Modules do not send messages directly to other modules but rather to "ports." A port is a local (to the module) name for one of the cables in the model -- thus modules are not aware of which other modules they receive messages from nor send messages to; they are aware only of their own local port names.

The patchboard is implemented as a set of "mailboxes" -- data structures which contain (among other things) a (possibly empty) set of messages. Patchboard connections are accomplished by making the "port information" portion of a process's DIB reference a particular mailbox.

A MESSAGE



MAILBOX



The message handling primitives are:

send (m,p) A copy of the message whose base address is 'm' will be sent to the mailbox connected to port 'p'. If the mailbox is currently full the sending process is suspended until space for the message becomes available.

receive (p) Return the address of a message in the mailbox connected to port 'p'. The message is removed from the mailbox. If no messages are currently in the mailbox the process is suspended until a message is sent to it.

The primitives and data structures for the kernel described above are defined precisely by the following Bliss module. This module was built for, and tested on, the PDP-10, but is identical to the PDP-11 version with three exceptions:

1. The full 36-bit PDP-10 word is used.
2. i/o for tracing and error reporting use PDP-10 monitor facilities.
3. The system function 'createprocess' will be somewhat different on the PDP-11.

Sample output from the tracing facility has been appended.

```
MODULE SL230(STACK)=
BEGIN

!                       SL230  -- SOFTWARE LAB
!                       --------------------


! SYSTEM PARAMETERS
! -----------------

BIND    MEMSIZE=4096,
        PSTACKSIZE=128,
        MSGLIMIT=2,
        MAXPORTS=3,
        NUMMAILBOXES=64;


! SYSTEM TRACING DEFINITIONS
! --------------------------

BIND    TRACE=#777777;

FORWARD TRG,TRR,TRC,TRNC,TRL1,TRL2,TRD1,TRD2,TRS,TRSV,TRP,TRV,TRSND,
        TRREC,ERROR;

MACRO

TGET=   IF TRACE↑(-0) THEN TRG(.N,.R[BASEF])$,
TREL=   IF TRACE↑(-1)THEN TRR(.A)$,
TCOPY=  IF TRACE↑(-2) THEN TRC(.A,.B)$,
TNEWCOPY=IF TRACE↑(-3) THEN TRNC(.A)$,
TLINK1= IF TRACE↑(-4) THEN TRL1(.A,.H)$,
TLINK2= IF TRACE↑(-4) THEN TRL2(.H)$,
TDLINK1=IF TRACE↑(-5) THEN TRD1(.H)$,
TDLINK2=IF TRACE↑(-5) THEN TRD2(.R[BASEF],.H)$,
TSWAP=  IF TRACE↑(-6) THEN TRS(.F,.T)$,
TSAVST= IF TRACE↑(-7) THEN TRSV()$,
TP=     IF TRACE↑(-8) THEN TRP(.S)$,
TV=     IF TRACE↑(-9) THEN TRV(.S)$,
TSEND=  IF TRACE↑(-10) THEN TRSND(.M,.PRT)$,
TRECV=  IF TRACE↑(-11) THEN TRREC(.R,.PRT)$;


! OBJECTS
! -------


STRUCTURE POBJECT[I,P,S,J]=
        ! STRUCTURE FOR A POINTER TO AN OBJECT
        CASE .I OF
            SET
            (.POBJECT+.J)<.P,.S>;
            (@.POBJECT+.J)<.P,.S>;
            (@@.POBJECT+.J)<.P,.S>;
            TES;

MACRO   BASEF=0,0,36,0$,          ! NAMES OF FIELDS IN AN OBJECT
        WORD(Z)=1,0,36,(Z)$,
```

```
        LINKF=1,0,36,0$,
        SIZEF=1,0,8,1$,
        PRIORITY=1,0,8,1$,
        NWORD(Z)=2,0,36,(Z)$,
        NLINKF=2,0,36,0$,
        NSIZEF=2,8,8,1$,
        NPRIORITY=2,0,8,1$;

STRUCTURE VECTOR[I]=[I](.VECTOR+.I)<0,36>;

GLOBAL VECTOR SPACE[16];

BIND VECTOR SIZE =
        PLIT(1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,
                32768,65536);

GLOBAL VECTOR MEM[MEMSIZE];      ! ALL OBJECTS ARE IN MEM


! SPACE MANAGEMENT
!  ----------------


FORWARD LINK,DELINK,COLLAPSE;
GLOBAL ROUTINE GET(N)=
    ! GET AN OBJECT OF SIZE 2**N AND RETURN ITS ADDRESS
    BEGIN REGISTER POBJECT R;
    IF .N LEQ 0 OR .N GEQ 16 THEN 0 ELSE
        BEGIN
        IF .SPACE[.N] NEQ 0
            THEN R[BASEF]←DELINK(SPACE[.N])
            ELSE (R[BASEF]←GET(.N+1); COLLAPSE(.R[BASEF]+.SIZE[.N],.N));
         R[LINKF]←0; R[SIZEF]←.N; R[PRIORITY]←0; TGET; .R[BASEF]
        END
    END;


MACRO   REPEAT= WHILE 1 DO$,
        BASE(B,S)= (B AND NOT(.SIZE[S]))$,
        PARTNER(B1,B2,S)= ((B1 XOR B2) EQL .SIZE[S])$;


ROUTINE COLLAPSE(A,N)=
    ! RELEASE THE SPACE FOR THE OBJECT WHOSE ADDRESS IS .A
    BEGIN MAP POBJECT A; REGISTER POBJECT L; TREL;
    REPEAT
        BEGIN L[BASEF]←SPACE[.N];
        WHILE .L[LINKF] NEQ 0 DO
            IF PARTNER(.L[LINKF],.A[BASEF],.N)
                THEN (A[BASEF]←BASE(DELINK(.L[BASEF]),.N); L[LINKF]←SPACE[(N←.N+1)])
                ELSE L[BASEF]←.L[LINKF];
        RETURN LINK(.A[BASEF],.L[BASEF])
        END;
    END;


GLOBAL ROUTINE RELEASE(A)=(MAP POBJECT A; COLLAPSE(.A[BASEF],.A[SIZEF]));
```

```
! OBJECT MANIPULATION PRIMITIVES
! ------------------------------


GLOBAL ROUTINE COPY(A,B)=
     ! CREATE A COPY OF OBJECT A IN B
     BEGIN MAP POBJECT A:B; TCOPY;
     INCR I FROM 2 TO .SIZE[.B[SIZEF]]-1 DO
         B[WORD(.I)]←.A[WORD(.I)];
     .B[BASEF]
     END;

GLOBAL ROUTINE NEWCOPY(A)=
     ! CREATE A NEW COPY OF A AND RETURN ITS ADDRESS
     BEGIN MAP POBJECT A; TNEWCOPY; COPY(.A[BASEF],GET(.A[SIZEF])) END;



GLOBAL ROUTINE LINK(A,H)=
     ! LINK OBJECT A INTO ITS CORRECT PRIORITY POSITION IN LIST H
     BEGIN MAP POBJECT A; REGISTER POBJECT L, P; TLINK1;
     P←.A[PRIORITY]; L[BASEF]←.H;
     WHILE .L[NPRIORITY] GEQ .P AND .L[LINKF] NEQ 0 DO L[BASEF]←.L[LINKF];
     A[LINKF]←.L[LINKF]; L[LINKF]←.A[BASEF]; TLINK2; .A[BASEF]
     END;

GLOBAL ROUTINE DELINK(H)=
     ! DELINK THE FIRST OBJECT IN H AND RETURN ITS ADDRESS
     BEGIN MAP POBJECT H; REGISTER  POBJECT R; TDLINK1;
     R←.H[LINKF]; H[LINKF]←.H[NLINKF]; TDLINK2; .R
     END;


GLOBAL ROUTINE SWAP(F,T)=(TSWAP; LINK(DELINK(.F),.T));




! SEMAPHORES AND SYNCHRONIZATION
!-------------------------------


STRUCTURE PSEMAPHORE[I]= (@.PSEMAPHORE+.I)<0,36>;


MACRO   COUNT=0$,
        SHEADER=1$;

GLOBAL POBJECT FEASIBLE:LASTRUN;


! DIBS, SIBS, AND PROCESS STUFF
! -----------------------------


MACRO   PORT(P)=1,0,36,(9+2*(P))$,
        NAMEF=1,0,36,3$,
        STKPTR=1,0,36,6$;


GLOBAL POBJECT DIRECTORY;
```

```
! PROCESS fANI''HI.ATJO'< HOUTl'ES
!.....................-.....

r.LOHAL ROUTINE SAVSTART*
     ! PERFORM A CONTFXT SWAp IF TOP 1*)F TEAS, LIST IS NOT RUNNING
     ir .FF.ASIPLFCBASEF I *IEQ ,LASTRUNI:BASEF3 THEN
         REOIM TSAVSTi
          ! RE11EtoF.R Sin LOC OF NEXT •RUNNING' PROCESS
          ! RUSH RFGISTERS R^-R'j
          ! LASTRUMCSTKPTR>.R6I R**'.FEASIRLECSTKPTR31
          j POP HACK REGISTERS HS-RB
          IF(LASTRUN^,FEASIBLE) EQL 0 THEN ERROR(1)|
          FXCHJ ( .FEARIHLffSTKPTRJ))
          FNOJ

GLOBAL ROUTIME P(S>'
     ! OIJKSTRAS »P» OPERATION
     BEGIM MAP ''SEMAPHORE Si TPI
     IF ( S[COUMT].-.Sr.COUMT3-1) Lf.S 0 THEN
          (SwAP(FFA"I"LrC''ASEF3»SCSHEADER])1 SAVSTART<) )1
     END)

GLOBAL ROUTINE V(S)»
     ! OIJKSTRAS «V» OPERATION
     OEGIK MAP PSEMAPHO^E Si TVI
     IF (5CC0UNT>.SCC0UNTD*.1) LEO 0 THEN
          (SWAP(SCSHEAQER].FEASIBLECBASEF3)I SAVSTARTO )|
     EN01

I MAILBOXES
I.........

STRUCTURE PMARBOXC1] » (P.PMAILBOX*,I)<9!,36>|

MACRO    MUTEXsZs;,
         ACCT1VITYs.$,
         LIMIT«6S»
         HHEAOE^=^JI

GLOBAL VECTOR MAILBOvESCNIJMM4ILHOXES}J

t MESSAGE HANDLING ROUTINES
I     ------------------------------

ROUTINF MBB(P)=
     BFGIM REGISTER R»
     IF ,P LSS f> OR ,P GTR MAVPORTS THEN ERBOR(?) ELSF
          IF (R«-1FEASIRLEtPQBT(•) 1) I.^S £1 THEM FRROR(3) ELSF
```

```
            IF .R GTR NUMMAILBOXES THEN ERROR(4) ELSE
                .MAILBOXES[.R]
    END;


GLOBAL ROUTINE SEND(M,PRT)=
    ! SEND MESSAGE M TO THE MAILBOX NAMED BY CURRENT PROCESS'S
    ! PORT #PRT, BLOCK THE PROCESS IF THE MAILBOX IS FULL.
    BEGIN MAP POBJECT M, PMAILBOX PRT; TSEND;
    PRT←MBR(.PRT);
    P(PRT[LIMIT]);  P(PRT[MUTEX]);
    LINK(NEWCOPY(.M[BASEF]),PRT[MHEADER]);
    V(PRT[ACCTIVITY]); V(PRT[MUTEX])
    END;


GLOBAL ROUTINE RECIEVE(PRT)=
    ! GET THE FIRST MESSAGE FROM THE MAILBOX NAMED BY THE CURRENT
    ! PROCESS'S PORT#PRT AND RETURN THE ADDRESS OF THIS MESSAGE.
    BEGIN MAP PMAILBOX PRT; REGISTER R;
    PRT←MBR(.PRT);
    P(PRT[ACCTIVITY]); P(PRT[MUTEX]);
    R←DELINK(PRT[MHEADER]);
    V(PRT[LIMIT]);  V(PRT[MUTEX]);
    TRECV; .R
    END;


!  SYSTEM (NOT KERNEL) SUPPORT FUNCTIONS
! --------------------------------------


FORWARD LOG2;


ROUTINE INITIALIZE=
    BEGIN
    DECR I FROM 16 TO 0 DO SPACE[.I]←0; SPACE[LOG2(MEMSIZE)]←MEM<0,0>;
    DECR I FROM (MEMSIZE-1) TO 0 DO MEM[.I]←0;
    DECR I FROM (NUMMAILBOXES-1) TO 0 DO MAILBOXES[.I]←0;
    LASTRUN←-1; FEASIBLE←0;
    END;


ROUTINE LOG2(N)=
    INCR I FROM 1 TO 16 DO
        IF .SIZE[.I] GEQ  .N THEN EXITLOOP .I;


ROUTINE CONNECT(DIR,PRT,MB)=
    BEGIN MAP POBJECT DIR;
    IF .MAILBOXES[.MB] EQL 0 THEN
        MAILBOXES[.MB]←COPY(PLIT(0,0,1,0,0,0,MSGLIMIT,0), GET(3));
    DIR[PORT(.PRT)]←.MB;
    END;
```

```
MACRO CREATEPROCESS(PROC,NAME,PRIOR)=
    BEGIN REGISTER POBJECT R,P;
    R[BASEF]←GET(4);
    R[STKPTR]←CREATE PROC AT GET(LOG2(PSTACKSIZE)) LENGTH PSTACKSIZE THEN 0;
    R[PRIORITY]←PRIOR; R[NAMEF]←NAME; LINK(.R[BASEF],FEASIBLE);
    .R[BASEF]
    END$;


! PRIMITIVE I/O FUNCTIONS FOR PDP-10 USE
! ------------------------------------------


MACHOP TTCALL=#51;
MACRO OUTC(X)=(REGISTER Q; Q←(X); TTCALL(1,Q); 1)$,
      OUTS(X)=TTCALL(3,X)$,
      OUTB(Z)=(INCR I FROM 1 TO (Z) DO OUTC(" "))$,
      CR=#15$, LF=#12$, CRLF=(OUTC(CR);OUTC(LF))$, TAB=OUTC(#11)$;
GLOBAL ROUTINE OUTN(N)=
    BEGIN REGISTER R,L; L←0;
    IF .N LSS 0 THEN (N←-.N; OUTC("-"));
    IF .N EQL 0 THEN OUTC("0") ELSE N←.N AND #777777;
    R←.N MOD 8;
    IF (N←.N/8) NEQ 0 THEN L←.L+OUTN(.N);
    OUTC(.R+"0")+.L
    END;




! ERROR REPORTING ROUTINES
! ------------------------


    ! NOTE THE FOLLOWING ERROR NUMBERS
    !
    !
    !    1.  NO PROCESSES LEFT ON FEAS. LIST
    !    2.  PORT # IN SEND OR REC OUT OF RANGE
    !    3.  PORT NOT CONNECTED
    !    4.  ILLEGAL MAILBOX
    !


ROUTINE ERROR(N)=
    BEGIN MACHOP CALLI=#47; CRLF; CRLF; CRLF;
    OUTS( PLIT ('******',' ERR ','#'));
    OUTN(.N);
    CRLF; CRLF; CRLF; CRLF; CRLF;
    CALLI(1,#12);
    END;



! SYSTEM TRACING ROUTINES AND MACROS
! ----------------------------------
```

```
MACRO   OUTP(Z)=OUTS(PLIT Z)$,
        PFN=(OUTP('  P: ');OUTS(FEASIBLE[NAMEF]);TAB)$,
        PLN=(OUTP('P:  ');OUTS(LASTRUN[NAMEF]);TAB)$,
        OUT1N(Z)=(TAB;OUTN(Z))$,
        OUT2N(Z1,Z2)=(TAB;OUTN(Z1);TAB;OUTN(Z2))$,
        OUT3N(Z1,Z2,Z3)=(TAB;OUTN(Z1);TAB;OUTN(Z2);TAB;OUTN(Z3))$;

ROUTINE TRG(N,G)=(CRLF;PFN;OUTP('GET');OUT2N(.N,.G));
ROUTINE TRR(A)=(CRLF;PFN;OUTP('REL');OUT1N(.A));
ROUTINE TRC(A,B)=(CRLF;PFN;OUTP('COPY');OUT2N(.A,.B));
ROUTINE TRMC(A)=(CRLF;PFN;OUTP('MCPY');OUT1N(.A));
ROUTINE TRLST(H)=(CRLF;TAB;WHILE .H NEQ 0 DO (OUT1N(.H);H←..H));
ROUTINE TRL1(A,H)=(CRLF;PFN;OUTP('LINK');OUT2N(.A,.H);TRLST(.H));
ROUTINE TRL2(H)=(CRLF;TAB;OUTP('LNK2');TRLST(.H));
ROUTINE TRD1(H)=(CRLF;PFN;OUTP('DLNK');OUT1N(.H);TRLST(.H));
ROUTINE TRD2(A,H)=(CRLF;TAB;OUTP('DLN2');OUT1N(.A);TRLST(.H));
ROUTINE TRS(F,T)=(CRLF;PFN;OUTP('SWAP');OUT2N(.F,.T));
ROUTINE TRSV=(CRLF;OUTP(('*****','SAVST',' F:  '));PLN;OUTP(' T:  ');PFN);
ROUTINE TRP(S)=(CRLF;PFN;OUTP('P');OUT2N(.S,@@S-1));
ROUTINE TRV(S)=(CRLF;PFN;OUTP('V');OUT2N(.S,@@S+1));
ROUTINE TRSND(M,P)=(CRLF;PFN;OUTP('SEND');OUT3N(.M,.P,MBR(.P)));
ROUTINE TRREC(M,P)=(CRLF;PFN;OUTP('RECV');OUT2N(.M,.P));




! TEST PROGRAM FOR PDP-10 IMPLEMENTATION
! -------------------------------------


OWN T;

ROUTINE P1(N)=
    BEGIN
    LOCAL L; L←GET(3);
    WHILE 1 DO
        (SEND(.L,1);CRLF;OUTN(.N);RELEASE(RECIEVE(0)));
    END;


INITIALIZE();
T←CREATEPROCESS(P1(1),'PA',1);CONNECT(.T,0,0);CONNECT(.T,1,1);
T←CREATEPROCESS(P1(2),'PB',1);CONNECT(.T,0,1);CONNECT(.T,1,2);
T←CREATEPROCESS(P1(3),'PC',1);CONNECT(.T,0,2);CONNECT(.T,1,0);
SAVSTART();


END
ELUDOM
```

Example Trace Output

Below is an example of the output obtained when the full tracing mechanism is turned on. The first line shows that a context swap from a process named PA to one named PB has occurred. The subsequent lines contain the process name (PB) and the name of a kernel primitive which it is calling at the left; to the right values of the parameters and results of the function are printed. Thus, for example, the line

```
P:PB    GET     3       10130
```

indicates that the GET function has been called to request 2^3 words of storage and that GET has returned the address 10130.

```
*****SAVST F: P: PA       T:  P:PB
 P:PB   ELNK     7763
                 7763     10130
        ELN2     10130
                 7763
 P:PB   GET      3        10130
 P:PB   SEND     10130    1         10060
 P:PB   P        10036    1
 P:PB   P        10062    00
 P:PB   NCPY     10130
 P:PB   ELNK     7765
                 7765     10140
        ELN2     10140
                 7765
 P:PB   GET      5        10140
 P:PB   DEL      10160
 P:PB   LINK     10160    7764
                 7764
        LNK2
                 7764     10160
 P:PB   GET      4        10140
 P:PB   DEL      10150
 P:PB   LINK     10150    7763
                 7763
        LNK2
                 7763     10150
 P:PB   GET      3        10140
 P:PB   COPY     10130    10140
```

[1] Clark, W., "Macromodular Computer Systems," SJCC 67.

[2] Bell, G., et al., "The Design, Description and Use of DEC Register Transfer Modules (RTM)," Computer Science Department Report, Carnegie-Mellon University, Oct. 1971.

[3] Krutar, R., private communication related to his Ph.D. thesis, Carnegie-Mellon University, 1971.

[4] Jones, A., and Habermann, A. N., "Interprocess Communication Mechanism," Internal Memo, Computer Science Department, Carnegie-Mellon University, 1970.

[5] Wulf, et al., "Bliss Reference Manual" Computer Science Department, Carnegie-Mellon University, revised April, 1971.

[6] Dijkstra, E., "Cooperating Sequential Processes," Technological University, Eindhoven, 1965.

[7] Wirth, N., "Program Development by Stepwise Refinement," CACM, Vol. 14, No. 4, (April, 1971).

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Computer Science Dept. Carnegie-Mellon University Pittsburgh, Pa. 15213	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

A SOFTWARE LABORATORY PRELIMINARY REPORT

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Scientific Interim

5. AUTHOR(S) (First name, middle initial, last name)

Corbin, Corwin, Goodman, Hyde, Kramer, Werme, Wulf

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
Aug. 23, 1971	25	7

8a. CONTRACT OR GRANT NO. F44620-70-C-0107 b. PROJECT NO. A0827-5 c. 61101D d.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES TECH, OTHER	12. SPONSORING MILITARY ACTIVITY Air Force Office of Scientific Research 1400 Wilson Blvd. (SRMA) Arlington, Va. 22209

13. ABSTRACT

This report describes the implementation of the kernel of a simple multi-process operating system. The purpose of this system is to create an environment for the construction of experimental programming systems for educational and research uses.

DD FORM 1 NOV 65 1473

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT