THE MULTI-TASKER

Volume 16, Number 3

April 1983

The Newsletter of the RSX-11/IAS Special Interest Group

Table of Contents

Columns

READ THIS FIRST

This issue has several articles that should be of interest to everyone. Carl Mickelson has written a wrapup article on the BRU sorting problem that he has described in the past. A must article for everyone who uses BRU.

Ralph Stammerjohn has written the first of what will hopefully be a continuing stream of articles. First, is a dictionary of over 300 words, phrases, and acronyms related to RSX/IAS. Second, is an article describing the SYSGEN philosophy at Monsanto. If you must SYSGEN more than one system read this article carefully. Perhaps most important are some of Ralph's patches; included are patches that can recover up to 1400(8) bytes of pool.

From the Editor

I would like to personally thank Ralph Stammerjohn for the fine job he did as the editor of the MULTITASKER. I didn't realize how much work was involved in putting an issue together until I started putting this issue together (which explains why it is so long in coming).

The Anaheim Symposia is history. For those of you who were unable to attend the biggest announcement was that DECUS is going to have a subscription fee. Effective July 1, 1983 if you want to receive any newsletters you will have to pay for them. This fee has become necessary due to the dynamic growth rate of DECUS and inflation. The subscription fee has not been set as of yet, but we should be informed around March 1. DECUSCOPE will not be affected by the subscription fee. The only known fact at this time is that there will be a subscription fee, exactly what the fee will be or what it will include are not known for sure.

Since we will all be charged for the priviledge of receiving the MULTITASKER it is my intention to provide an issue every month of the highest possible quality. In order to provide this I need articles to publish and these articles come from you the readers. An article can be on any subject, even IAS, of any level of expertise. A large majority of our audience are new users. They need articles that many of us might consider everyday or simple. Articles should preferably be in machine readable format (tape or floppy) or of a publishable quality on 8 1/2 by 11 plain white paper.

Charles Goodpasture
Multi-Tasker Editor

Phone: (713) 871-8000

DECUS/RSX SIG Library News

Paul Tompkins
Library News Editor

Over the years, DECUS, through the DECUS library, and the RSX-11/IAS SIG, through the SIG tapes, have accumulated a huge set of useful software. If you have news about any of this software, please send to the Multi-Tasker c/o this column. This includes any problems discovered, patches to existing software, short notes on library submissions you found useful, or any other information you may have. Send submissions to Multi-Tasker - Library News, c/o DECUS, One Iron Way, MR2-3/E55, Marlboro, MA 10752.

DIR

David WeBlatt

The University of Newcastle
Department of Mathematics, Statistics and Computer Science
New South Wales, 2308

I would like to advise on the following problems with "DIR" and "RNO", both of which I have found very useful.

DIR

When used without the "V" switch DIR should list the latest versions of a file in the directory. As version numbers are not included in the printout, the date and time are the only evidence. Sometimes, it seems, DIR gets these wrong. Presumably, it is picking up the first occurence in the directory(?).

DIR will not wildcard on directories [*,*] so that it becomes awkward to get sorted directories of complete volumes. I have enclosed a cumbersome but effective solution to this using a command file, DIRALL.CMD. Basically, it does a PIP [0,0]*.DIR/LI to get the list of directories on the volume, then generates and executes a command file called DIRALL.TMP, which it finally deletes. Perhaps someone intends to enhance DIR to provide some decent wild carding, but meanwhile, this may be of use.

DIRALL.TEC

```
        @i/en/ zj @i/[0,0]*.DIR$/@i/$/
        hxb hk
        mb@EWDIRALL.TMP$
        EN$i.ENABLE SUBSTITUTION
        $
        <:EN$;G*OLI;
        .OPEN NAME.TMP
        .ENABLE DATA
        '$BLANK'
        ==========='<DATE>' '<TIME>'===============
        Directory listing of '$PAK'
        dir >tx: -lev $
        .ub s[$ .ua s]$ qa,.k 3c i,$ 3c i]
        $.,zk qb,.xa -1 rr ga
        1 '$blank'
        .disable data
        .close
        PIP TX:=NAME.TMP
        PIP NAME.TMP;*/DE
        $ZJ>
        EX$$
```

DIRALL.CMD

```
        ; USE DIR UTILITY TO MAKE A COMPLETE SORTED
        ; DIRECTORY OF A DEVICE, ALL UIC'S
        .; DWEB 12-AUG-82
        .ENABLE SUBSTITUTION
        .ENABLE GLOBAL
        .SETS $BLANK "  "
        .ASKS $PAK PACK IDENTIFICATION STRING
        .SETS DFALT "DK"
        .ASKS DEV DEVICE TYPE ['DFALT']
        .IF DEV = "" .SETS DEV DFALT
        .ASKS DVN 'DEV' UNIT NUMBER
        .ASKS OUT OUTPUT TO [DEFAULT TI:]
        .IF OUT = "" .SETS OUT "TI:"
        ASN 'OUT'=TX:
        PIP TX:=DK7:[1,2]FORMFEED
        MUNG DK7:[1,2]DIRALL,'DEV''DVN':
        @DIRALL.TMP
        PIP DIRALL.TMP;*/DE
        ;
        ; END OF DIRECTORY LISTING OF 'DEV''DVN': TO 'OUT'
        ; LISTING IDENT IS '$PAK'
```

RNO

An awkward prblem with RNO is that linefeeds, by themselves, are not included in the count of lines on a page. It is sometimes necessary to have a carriage return - linefeed pair. This can make literal sections (.lit to .eli) give rise to embarassing page overflows. Files transferred from other systems (eg. in FLX RT11 format) often have bare linefeeds for multiple blank lines, giving rise to this problem. It is necessary to either write a filter program or do a messy edit run (e.g. with teco and controls visible on a VT100 by -1,3:w) to name.RNO files.

SYSLOG

James G. Downward

KMS Fusion, Inc.
P.O. Box 1567
Ann Arbor, MI 48106

It recently came to my attention that a fatal "BUG" exists in SYSLOG for RSX11M V4.0 for those users who include XDT in their system. Conditional assembly code to handle source code only used by XDT was not included in the distribution source. If SYSLOG is run and the user selects QIO accounting, the system will promptly CRASH. The enclosed SLP file solves this problem. I apologize for not catching this problem sooner. Since we never use XDT on a system which runs accounting, I never noticed the problem until a user who generated XDT in their system called up to ask why his system crashed.

To solve the problem (if you use XDT), rename SYSLOG.MAC to SYSLOG.VGN and create the following SLP correction file, SYSLOG.SLP.

```
SYSLOG.MAC/AU=SYSLOG.VGN

-2,2
        IDENT   /04.1/
-/V4.0 compatible/,,
;
;       JDG4        Fix bug in $IOFIN intercept address
;       19-Sep-82   would crash systems with XDT
;
%
-/QADTB:/,.,/; JDG4/
QADDTB:
        .IFDF   A$$TRPR$$DER        ; With XDT in system

        .WORD   $IOFIN+12           ; 1, I/O COUNT INTERCEPT AD-
DRESS
        .IFF                        ; with no XDT
        .WORD   $IOFIN              ; 1, I/O COUNT INTERCEPT AD-
DRESS
        .ENDC
/
```

Install SLP and type SLP @SYSLOG.SLP. Then proceed to use SYSLOG.CMD to rebuild SYSLOG.

Working Group News

Elizabeth Bailey
Working Group News Editor

BUG in INSTALL for RSX-11M

Recently Robert Gezelter encountered a bug in INSTALL on RSX-11M Versions 3.2 and 4.0.

The problem was that in certain case a block of the task's header must be written to disk while INSTALL is processing COMMON/LIBRARY region references. Unfortunately the code in module INSLB which processes the COMMON/LIBRARY references uses R0 to hold the PCB pointer for the common region. Subroutine $RDHDR (which is called by INSLB) will destroy R0 if the current header block is full. This inconsistent handling of R0 by $RDHDR is fixed by saving R0 upon entry to $RDHDR and restoring R0 upon exit.

RSX prints the following error message when this condition occurs:

```
        Base mismatch in common block xxxxxx
```

The patches for the various versions of RSX-11M are as follows:

Version 3.2

```
-2,2,/;RLG001/
        .IDENT  /04.1/
-39,,/;RLG001/
;       RLG001 09-AUG-82  FIX BUG IN $RDHDR WHICH
;                         SOMETIMES DESTROYS R0
-229,229,/;RLG001/
$RDHDR::MOV     R0,-(SP)  ; SAVE R0 ON THE STACK
        CMP     R4, $HDRBF+512. ;HEADER BLOCK FULL?
-244,244,/;RLG001/
30$:    MOV     (SP)+,R0  ; RESTORE R0 FROM THE STACK
        RETURN            ; RETURN TO CALLER
/
```

VERSION 4.0

```
-2,2/;RLG001/
        .IDENT  /06.1/
-47,,/;RLG001/
;       R. GEZELTER - FIX BUG IN $RDHDR WHICH
;                     SOMETIMES DESTROYS R0
-256,256,/;RLG001/
$RDHDR::MOV     R0,-(SP)
        CMP     R4, $HDRBF+512.
-271,271,/;RLG001/
30$:    MOV     (SP)+,R0
        RETURN
/
```

William D. Burton Jr., Chairman of the Unsupported Versions Working Group, reports that the same bug appears to be

in Versions 3.0 and 3.1. The symptoms appear to be either "BASE ADDRESS MISMATCH..." or "PARAMETER MISMATCH..." error messages.

The correction file for [12,10]INSRO1.MAC for RSX-11M Versions 3.0 and 3.1 is as follows:

```
-2,2,/;RLG001/
        .IDENT  /03A/
-37,,/;RLG001/
;       RLG001 09-AUG-82 FIX BUG IN $RDHDR WHICH
;                        SOMETIMES DESTROYS R0.
;                        (ADAPTED FOR V3.0 AND V3.1)
-215,215,/;RLG001/
$RDHDR::MOV     R0,-(SP)
        CMP     R4, $HDRBF+512.
-230,230,/;RLG001/
30$:    MOV     (SP)+,R0
        RETURN
/
```

The Virtual Disk Working Group met in Anaheim at the RSX SIG general working group session. Current active versions of the virtual disk packages were discusses. It was decided that the primary goal of the group at this time is to consolidate the various features into a complete software package with documentation and command files. The code should be conditionalized for RSX-11M and RSX-11M+ and upgraded to the current version level. This package should then be field tested by members of the working group. On completion of software checkout, the complete package will be submitted to the RSX SIG tape and the DECUS library. At that time, the working group will address extensions and enhancements to the software package.

Any suggestions or enhancements that you have or would like to see implemented should be mailed to the Chairman of the Virtual Disk Working Group:

Mr. Robert Hayes
Union Carbide Corporation
P. O. Box 3500
Oak Ridge, Tennessee 37830

Help Yourself

Help Yourself

David DiGiacomo
Help Yourself Editor

"Help Yourself" is a place for you to get your tough questions answered. Each month, questions from readers will be published. If you have a question, send a letter to the Multi-Tasker - Help Yourself, c/o DECUS, One Iron Way, MR2-3/E55, Marlboro, MA 01752.

We would also like to publish the answers to questions. If you can help someone, contact the Multi-Tasker. Your answer will be sent directly to the person in need and published in the next edition.

This Month's Questions

RT Emulator

We are trying to locate an RT-11 emulator to run under RSX-11M V3.2. MINITAB under RSX-11M requires a FP11 floating point unit since the taskbuilder (RSX-11M SLOTKB) can not cope with MINITAB unless FP11 is installed. The RT-11 linker can, on the other hand, link MINITAB without a FP11. We are aware of RTEM-11 from DEC but are looking for a more economical solution.

Friorik Marteinsson, Hus verslunarinnar, 108 Reykjavik, Iceland

Hewlett Packard 3354

I have a PDP 11/45 running RSX-11D and I am trying to communicate with a Hewlett Packard 3354 laboratory system via a direct link. If anyone has successfully done this, please contact me at your convenience.

Randolph P. Brown, U.S. Environmental Protection Agency
9311 Groh Road, Grosse Ile, Michigan 48138

Fun Programs

We are new users with a PDP-11/23, RSX-11M V3.2 with FORTRAN 4, using 2 RL02 disks. We are non-programmers at a branch office and would like to obtain print-outs of some of the smaller fun programs that we can install ourselves.

James R. Lee, Barringer Resouces
1455 Deming Way, No. 15, Sparks, Nevada, 89431

Does anyone have generalized REGIS routines to drive a VT125 from RSX-11M?

B. J. Checkowy, Saskatchewan Telecommunications
2121 Saskatchewan Drive, Regina, Saskatchewan S4P 3Y2

DQ11 Driver

Our new hardware configuration requires that our PDP 11//44s interface with up to four full-duplex serial synchronous communications lines simultaneously. Investigations into available interface boards have located only one, Digital's DQ11, that is DMA and operates at the required baus rates (7.2 to 56 KBaud).

Does anyone know of any cost-effective alternatives to the DQ11 interface boards that are DMA and operate in the 7.2 to 56K Baud range?

We are utilizing DEC's RSX-11M Version 4.0 operating system. The DQ driver software under this operating system is strictly half-duplex. Has anyone been able to modify the existing DQ driver software or develop stand-alone software that permits the DQ!! board to be operated in the full-duplex mode?

Raylene Pak, Bendix Field Engineering Corporation
P.O. Box 67, Moffett Field, California, 94035

Continuous Flow Analyzers

We are runnig RSX-11M on a PDP 11/34 to collect data from up to 32 laboratory instruments, either continuous flow analyzers or HPLC's. The data collection system is working well, but a current problem is the selection of peaks from the raw data. The standard DEC laboratory subroutines package is perfectly adequate for c.f. analyzers, but less than satisfactory for the more complex peaks from the HPLC's. There are only three peak-picking parameters available, compared with a minimum of ten for most commercial integrators and the determination of peak hights and areas is often inadequate.

I am sure there must be some better PDP-11 compatible software available for GC's or HPLC's but I cannot find any in the U.K. Do you know of any in the U.S.A., or if any of your SIG can help. I would be grateful.

C.B. Taskis, Beecham Pharmaceuticals
Clarendon Road, Worthing, West Sussex BN14 8QH

It's in the Code

Jim Preciado
Column Editor

FORTRAN IV-Plus V3.0

D. Scott Campbell

Par Technology Corporation
4575 Hilton Parkway, Suite 200
Colorado Springs, CO. 80907

The FORTRAN IV Plus compiler has several undocumented switches which produce some interesting, but probably not very useful, variations on the listing obtained, and on the level of compiler optimization. They are used as follows:

SWITCH	EFFECT
/FI:2	Produces an internal dump of some sort, perhaps the symbol table. Not sure about effects other than values for the switch.
/LI:n	ORing 10(8) to the /LI: switch produces a dump of the compiler internal data structures and pseudo object code. For ex /LI:13 will produce the normal listing (source + p-sections + symbols + generated macro) as well as the dump.
/OP:n	Alters the level of optimization used by the compiler. 0 is no optimization, 3 is the highest (default).

Aside from given the curious something to look at, I can see very little use for the dumps produced by the two switches. However, if you suspect the compiler is making an error due to improper optimization, specifying no optimization (/OP:0) will allow you to make this determination.

EDT V2.0

Mike Kabo

University of California, Los Angeles
1000 Veterans Avenue
Los Angeles, CA. 90024

For what it's worth, these are just a few lines to inform users of EDT V2.0 of an undocumented feature in keypad mode which may or may not prove useful. Use of the gold + "left arrow" or "right arrow" in keypad mode shifts the screen window contents to what appears to be one tab stop (8 spaces) to the left or right, respectively. Neither the set tab command nor alteration of the hardware tabs seems to have any effect on altering this increment. This action only affects the viewing window and not the contents of the file, i.e., inclusion of characters either from the keyboard or from a buffer are inserted beginning at the true cursor position.

SYSLOGIN and SYSLOGOUT Command Files

Allen A. Watson

The Record
150 River Street
Hackensack, NJ 07602

When doing a SYSGEN it is hard to avoid discovering the fact that RSX-11M+ V2.0 supports a [1,2]SYSLOGIN.CMD file. It is not at all obvious, unless you happen to read the MCR manual on the BYE command, that it also supports a [1,2]SYSLOGOUT.CMD file. (And who thinks they need to read about "BYE?) Basically, if there is a file called [1,2]SYSLOGOUT.CMD, BYE will execute it before signing you off.

In the MCR manual under HELLO is a sample SYSLOGIN.CMD file showing how to make it chain to a user's LOGIN.CMD file. We put a command in ours to notify users if they had mail (we are using the MAIL program from KMS FUSION currently and experimenting with the one from the LBL tools package).

Using a similar concept I created a SYSLOGOUT.CMD file that also chains to a user's LOGOUT.CMD file. It also notifies of unread mail. Here is ours:

```
 .ENABLE SUBSTITUTION
 .ENABLE QUIET
 .TESTDEVICE TI:
 .SETS TI <EXSTRI>[1:2]
 .IF TI = "VT" .GOTO BATCH
 MAIL?
 .DISABLE QUIET
 .SETS FILE <LOGDEV>+":"+<LOGUIC>+"LOGOUT.CMD"
 .TESTFILE 'FILE"
 .IF <FILERR> = 1 .CHAIN 'FILE'/LO
.BATCH: .EXIT
```

I also made the file check to see whether it is a batch (virtual) terminal logging out, and if so did nothing.

When I was testing my LOGOUT.CMD file, I put a PIP /FU directory command in it. Lo and behold, after about 10 seconds, both AT and PIP were aborted by BYE just as if they were running when I logged off. Yet it did word for a while. I figured there must be a timeout parameter somewhere in BYE. Found it, called INDTIM. Looked in the task build file, [1,24]BYEBLE.CMD, and found it is a task build option, set to octal 12 seconds by default. I decided a three minute timeout might provide enough time to do something useful at logout and so changed it to 264 octal (180 decimal). Rebuilt BYE and it work's just fine.

There are some other interesting GBLDEF's there in BYEBLD. Here they are:

```
GBLDEF=$USRSB:0 ; ADDR OF USER SUBROUTINE (0=NOT USED)
GBLDEF=$MALSB:0 ; ADDR OF MAIL NOTIFICATION SUBROUTINE
                ; (0=NOT USED)
GBLDEF=INDTIM:264 ; TIMEOUT FOR INDIRECT (SECONDS)
GBLDEF=ABOTIM:0 ; TIMEOUT FOR ABOURT AST (SECONDS)
```

I intend to investigate the code in BYE.MAC to see what possible use might be made of the USER SUBROUTINE AND MAIL NOTIFICATION SUBROUTINE options, but meanwhile I was wondering if anyone else has already experimented. Anybody?

EDITORS NOTE

When testing the SYSLOGIN.CMD file ensure that there is another privileged CRT logined to the system. If you make a mistake in the command file and AT aborts your terminal will be set SLAVED. Also keep a copy of the command file in another UIC so that if you have problems with the file the version in [1,2] can be deleted to allow other users to login.

Hints And Things

"Hints and Things" is a monthly potpouri of helpful tidbits and rumors. Readers are encouraged to submit items to this column. Any input about any way to make life easier on RSX/IAS is needed. Please beware that items in this column have not been checked for accuracy. Send any contributions to Multi-Tasker - Hints and Things, c/o DECUS, One Iron Way, MR2-3/E55, Marlboro, MA 01752.

Preserving Resident Data Common

Phil Rowland

Celanese Corporation
Charlotte, N.C.

An efficeint way to keek large applications synchronized is to keep pointers and flags in a resident data common. A resultant problem is how to keep the disk image (nearly) current. One way is to update the disk image. But why have a resident image.

This article describes how to periodically, and especially during SHUTUP, update the disk image. This procedure does not require creating new files and the updated data will be loaded at the next Reboot or Install.

Create the resident data common image as described in the Task Builder manual. Carefully note the first data disk block from the map, normally Block Group and is used to Install to properly load the image. This area should not be disturbed and is the key to this procedure.

Create a program which fully describes the common data and links to it. Use equivalences for clarity. Include the following (FORTRAN) code:

```
      OPEN (UNIT=dd, NAME='[uic]name.tsk', TYPE='OLD',
      1 ERR=xxx, FORM='FORMATTED', SHARED)
      READ (dd,1010)                      !SKIP LABEL GROUP
1010  FORMAT(2(512X))                     !TWO BLOCKS WORTH
```

```
      WRITE (dd,1020) KOM                 !WRITE COMMON
1020  FORMAT (adjust for data type and length)
      CLOSE (UNIT=dd)
```

Run the program with some appropriate reschedule interval, possibly one hour. Also include its execution in SHUTUP.CMD.

Tricks with FMS-11

John P. Hoekstra
Yasunobu Suginaka
powell Quiring

Macatawa Computer Services, Inc.
1 West Fifth Street
Holland, MI 49423

We recently decided to use FMS for a real time display application. The named fields and shared forms features seemed useful. There was only one problem, FMS does not support dynamic (run time) changing of video attributes (blinking, etc.). We require this to alert the user to any out of range parameters.

The way around this problem that we are using is to define the fields that will have varying attributes with no video attributes and then define a separate disply only field in a blank part of the screen that allows any ASCII characters (we call this field ATTRIB). The trick is that FMS only writes the escape sequences for setting the attributes of fields so defined. The video terminals associate the escape sequence with the following character stream. So if you put out the required escape sequence for setting video attributes to the ATTRIB field any field after the put will be displayed on the terminal with the requested attributes set. After you have finished writing the fields you will need to put the reset attributes escape sequence to the ATTRIB field.

You can take this idea one step further by storing the printable parts of the escape sequence for setting and resetting video attributes in the named data area of each form. The application program can then get the field called BLINK from the form, add an escape character to the beginning, put the resulting string to the field ATTRIB, put out any field requiring blinking, get the field called RESET and do the same put to ATTRIB.

The advantage of this approach is that the task independent features of FMS are preserved (ie. change the forms layout without relinking). The disadvantage is that FMS doesn't restore them if a screen refress is done.

CINT$ (Switch to System State)

Terry Coombes

Logica (Benelux)BV
Vasteland 12
3011 BL
Rotterdam, Nederland

The size of a privileged task mapped onto the I/O page and a 20K RSX-11M executive is limited to 8K words. Very vew privileged tasks need to be mapped onto the exec at all times during their execution. Indeed, it is desirable that the exec be protected from such tasks as much as possible. This contribution allows a privileged (PR:0) task to make use of the full 32K virtual address space for its own code while retaining the ability to map onto executive data structures and code whenever necessary.

The following macro and subroutine exploit the "interrupt-enable" feature of the Connect-to-Interrupt directive to switch the task into system state. Once in system state, the task has acces to the I/O page, the executive, and 4K words of its own code/data, starting at the macro call. Code executed in system state must obey the rules set out in the documentation for the CINT$ system directive. The task returns to task state by executing an RTS PC instruction.

Apart from the need to ensure that support for CINT$ has been included during Sysgen, the only prerequisite is an unused vector doubleword (preferably) near the top vector area in low memory.

```
        .MACRO  SWSTK,RET,ERR,?LAB
;
        JSR     R0,SWSTK          ; ISSURE CONNECT DIRECTIVE
        RETURN                    ; DUMMY INTERRUPT SERVICE ROU-
TINE
        .WORD   ERR               ; ADDRESS OF ERROR PROCESSING
CODE
        .WORD   RET               ; CODE ADDR AFTER RETURN TO
```

```
TASK STATE
        CLR     X.DSI(R1)         ; ENSURE NO CALL ON DISCONNECT
        CALL    LAB               ; PERFORM USERS'S SYSTEM-STATE
CODE
        JMP     @ $DISIN          ; FORCE IMMEDIATE DISCONNECT
LAB:
;
        .ENDM


        .MCALL  CINT$,DIR$,ITBDF$
        ITBDF$
        .GLOBL  X.DSI,X.TCB
CINT:   CINT$   0,0,0,0,0,0
;
SWSTK::
        MOV     R0,CINT+C.INBA    ; BASE IS IMMED. AFTER SWSTK
        MOV     R0,CINT+C.INIS    ; SO IS DUMMY INT SERVICE RTN
        ADD     6,R0              ; SKIP ISR AND USER RET ADDR
        MOV     R0,CINT+C.INDI    ; ADDR OF SYSTEM-STATE CODE
        MOV     $V$$CTR,CINT+C.INVE     ; VECTOR IS JUST LEGAL
10$:    SUB     4,CINT+C.INVE     ; POINT AT NEXT VECTOR DOWN
        DIR$    $CINT             ; TRY TO CONNECT TO IT
                                  ; SYSTEM-STATE CODE ALREADY -
                                  ;  EXECUTED IF SUCCESSFUL
        BCC     20$               ; BACK TO CALLER IF FINISHED
        CMP     CINT+C.INVE, 60   ; IF NOT, CHECK IF ALL TRIED
        BGT     10$               ; TRY ANOTHER IF NOT
        TST     -(R0)             ; SKIP OVER SUCCESS RET ADDR
20:     MOV     -(R0)             ; GET SUCCESS/ERROR RET ADDR
        RTS     R0
```

Swedish Pascal under RSX V4

Seved Torstendahl

Telefon AB LM Ericsson
S-126 25 Stockholm, Sweden

The "Swedish Pascal" compiler will not run if built under RSX V4. The problem is that the compiler is heavily overlaid and that a patch must be applied to the overlay routine because of the recursive use of procedures in Pascal.

For RSX V4 the following changes have to be made:

- modify module RECURS in PASLIB.OLB using the following procedure:

1. extract RECURSEG.MAC from SRCFIL.PAC using the utility select or an editor.

2. apply the following SLP file to it:

```
        -    3
                .IDENT  /RSX.V4/
        -  17,     17
        ; $AUTO+36 := BR SEGENT
        -  23,     23
                MOV       SEGEXIT,14(SP)          ;RSX.V4
        -  28,     30
                JMP      $AUTO+42         ;READ SEGMENTS WHEN
READY
        ;                                 ;CONTROL TRANSFERRED
TO SEGEXIT
        99$:    JSR      R4,$WRERROR
        -  40,     41
                JSR      R5,.SAVR1
                MOV      @ N.OVPT,R1      ;NORMAL SEGMENT READ
START
        -  47,     49
                JMP      $AUTO+42
        ;
        9$: ;;  MOV      (SP)+,R2
        /
```

3. assemble: MAC RECURSEG=RECURSEG

4. modify PASLIB: LBR PASLIB/RP=RECURSEG

- change patch in PASLIB.CMD fron 34:435 to 36:441 giving the line

```
        GBLPAT=ROOT:$AUTO+36:441    ; BR    SEGENT
```

After these two changes the compiler can be rebuilt using TKB.

Speak Out

"Speak Out" is a monthly column for readers to express their opinions or to comment on a previous column. The articles published in this column are an individual's viewpoint and do not necessarily reflect the opinion of DECUS or the RSX/IAS SIG. Readers are welcome to submit articles on any subject concerning the RSX/IAS world. Send submissions to Multi-Tasker - Speak Out, c/o DECUS, One Iron Way, MR2-3/E55, Marlboro, MA 01752.

A Further Note on RX03 Problems

John G. Roth

University of Toronto Computing Services
Toronto, Ontario, Canada M5S 1A1

We also use DSD-880 floppy/Winchester on some of our systems and are disappointed about DEC's quiet decision to drop double sided support. I was also able to patch the DY: by comparing V3.2 and V4.0 source, but could not get the system to MOUnt double sided diskettes. I compared my patch with the patch given by Greg Liverman in the Sept/Oct Multitasker and found it to be the same (except for cosmetic differences).

The key, which Greg mentions with respect to the executive corruption problem, is in forcing MOUnt to do a sense characteristics QIO. (It appears that MOU is trying to read a block above the single side limit, in addition to reading block 1. MOU detects incorrect volume labels, but returns IE.RER errors if you specify the correct label or /OVR.). Not wnating to modify MOU, I wrote a 'one-liner' to do the sense and RUN it with a double sided diskette in the drive before MOUnting. Thanks to Greg for pointing the way; and I've sent DEC and SPR suggesting that MOU be fixed.

RSX Buzzwords

Ralph W. Stammerjohn

Monsanto
800 N. Lindbergh
St. Louis, Missouri 63167

The following is a alphabetical list of over 300 words, phrases, and acronyms related to RSX/IAS. The list covers a wide variety of subjects, from the very basic to executive internals. The definitions are given in the RSX/IAS context and not as they may globally apply to computers.

Many RSX terms have acronyms which are their more familiar form. This document has the definition listed under the acronyms and points to term to the shorthand description.

For further information on RSX/IAS concepts and basic information, the reader is refered to the Introduction to RSX-11M and RSX-11M-Plus (AA-L763A-TC) and the RSX-11M Information Directory and Index (AA-2593F-TC). These two manuals will get you started with the rest of the RSX manual set.

[0,0]
: [0,0] is the notation for the master file directory. [0,0] is the directory that has entries for the user directories. Each user directory shows up as a file with the name gggooo.DIR;1 - where ggg is the owner number and ooo is the group number. For example, directory [7,114] is a file in [0,0] with a name of 007114.DIR;1. Other system files are also found in [0,0]. These include the index file (INDEXF.SYS), disk block bit map (BITMAP.SYS), bad block file (BADBLK.SYS), and account file (RSX11.SYS). [0,0] does not normally show up in wildcard searches, but it can be directly named.

Abort
: Abort means terminating a task abnormally. This can either occur because the task had an unexpected failure due to a programming error (SST), another task issues the executive directive ABRT$, or a user from a terminal explicitly aborts the task. When a task aborts, the RSX-11M executive cleans up all the context for the task as best it can. See also I/O Rundown and SST.

Account
: Each RSX user is identified by a pair of octal numbers and a ASCII name and must use either the number or name to gain access to the system. These numbers and name are called accounts. The two octal account numbers are called the group and owner number. Each has a range of 1 to 377.

: When you log in, the RSX system program HELLO uses either the account number or name to find a record in the system account file. This file has a password which must match your supplied string and various defaults for your session. Your account number informs the system what initial directory you want to work from and what kind of access to files and other system facilities you will have. The account numbers are directly tied into

the protection mechanisms of RSX. Privileged users (those having group numbers equal or less than 10) have access to all system commands and all parts of the system. Nonprivileged users are limited to everyday operations that do not threaten the integrity of the system. See ACNT, File Protection, and Privilege User.

ACNT
: RSX accounts are kept in a special system file named LB:[0,0]RSX11.SYS. ACNT is the RSX system utility for defining, deleting, changing, and listing the current accounts in this file.

ACP
: ACP is an acronym for Ancilliary Control Processor. An ACP is a privilege RSX task with that is used to implement protocols for classes of devices. ACP's form an integral part of the RSX I/O mechanism. Examples of ACP tasks include F11ACP for the Files-11 disk and MTAACP for ANSI labeled magtapes. See I/O mechanism.

ACS
: ACS is the RSX system utility for defining and removing checkpoint files. See Checkpointing.

Active Task List
: See ATL.

Address Page Register
: See PAR.

Ancilliary Control Processor
: See ACP.

APR
: See PAR.

ASCII
: ASCII stands for American Standard Code for Information Interchange and is the standard coding used in RSX for character information. ASCII defines a 128 character set, which is stored in a PDP-11 in eight-bit bytes.

ASECT
: Memory allocation for various code segments when a program is linked in established by attributes assigned to program sections. ASECT is a special form of the general program section (PSECT) directive that defines an absolute segment. This tells the task builder that the segment cannot be relocated and must be linked starting at absolute virtual address zero. See PSECT.

AST

AST is an acronym for Asynchronous System Trap. A system trap is a transfer of control that is caused by some event. RSX defines two types of system traps, synchronous (SST) and asynchronous (AST).

An AST is a trap that occurs asynchronous to program execution. That is, the program has no direct control over the precise time and point the event - and therefore the trap - will occur. Examples of AST's include I/O completion, mark time expiration, or message sent from another task.

A program specifies AST notification for different types of events. When the event actually occurs, the executive queues the AST to the task. The AST will then interrupt the normal task execution the next time the task is scheduled and control will be passed to the AST trap address. The executive knows when a task is executing an AST and waits until the processing of the AST is complete before initiating another queued AST trap. A program can also selectively enable and disable AST notification.

Asynchronous System Trap

See AST.

ATL

ATL stands for Active Task List. The ATL is a priority-order list of all tasks actively competing for system resources, i.e. in execution. This includes tasks which may be checkpointed out of memory. See STD and TCB.

Attach

Attach has two meanings in RSX. First, attach is an I/O function (IO.ATT) which grants a program exclusive access to a non-shareable device. Once a program successfully attaches to a device, only its I/O requests will be sent to the device. All requests from other tasks, including their attach requests, will remain in the device queue.

Second, attach is a PLAS directive function that allows a task to gain access to a region of memory. Once a region is attached by a task, it can then map windows into the region and directly address it. See PLAS.

Attachment Descriptor

Attachment Descriptor is an RSX executive data structure stored in pool. It ties tasks to the various regions and partitions in memory that they have attached.

Autoload

Autoload is the method of loading overlay segments, in which the Overlay Run-Time routines automatically load overlay segments when they are needed. The asterisk (*) is the control symbol in overlay descriptor files that tells the taskbuilder to setup autoload vectors for the module.

Autoload Vectors

A transfer of control table generated by the Task Builder to resolve a overlay reference. When an overlaid routine is called, the reference is actually to the autoload vector which passes control to the Overlay Run-Time routines. These routines do any disk I/O necessary to bring the routine into memory and then pass control to the actual routine.

Autopatch

Autopatch is a part of the Software Maintenance Service mechanism. On a irregular basis (about three to four times a year), Digital makes a machine-readable compilation of all patches that apply to the current release of the operating system and many layered products. The media is typically magtape.

Autopatch kits are named alphabetically by their occurence: Autopatch A, B, C, and so forth. There is master documentation for the entire kit and specific documentation for each set of patches to layered products.

The patches supplied by Autopatch usually come in two forms: SLP correction files for distribution sources and PAT source files for distribution object modules and object libraries. The patches are always applied to the original distribution, no manner how many Autopatch kits may have been distributed in the past. Also, Digital has used the Autopatch kits to distributed completely new versions of software that did not make the original distribution.

Autopatch kits are historically not entirely bug-free and usually do not have a one-for-one relationship to the Software Dispatch.

BAD

BAD is a RSX system utility for detecting and recording the bad blocks on a disk. BAD writes a worst-case pattern to all blocks on a disk and reads the pattern back. Any errors detected are stored in the last good block on the disk where it is used by the disk initialization program, INI. BAD is also handles disks with last-track information. See Bad Block File.

Bad Block File
The bad block file is a RSX system file ([0,0]BADBLK.SYS) created when the disk is initialized by INI. The file contains all the bad blocks found on the disk by the BAD utility or from the manufacturers bad block data. Placing the bad blocks into a file makes them unavailable for allocation by other files.

Bitmap
A bitmap in RSX is a free/used mapping where each bit in the set indicates whether a item is in use or free for allocation. The bit is set on if the corresponding item is free. The mapping within a word is from right-to-left, i.e. bit 0 of the first word maps the first item, bit 1 maps the second item, and so forth.

Two examples of bitmaps are the index file bitmap and the disk allocation bitmap. The first indicates free/used blocks in the index file. The second is for block allocation for the entire disk.

Block
A block is a unit of measurement for disks. In RSX, a block is 256 words (512 bytes) and is the smallest addressable unit of a disk.

The term block is also used to refer to an arbitrary number of contiguous bytes used to store logically related information, such as a Unit Control Block (UCB), File Control Block (FCB), or Task Control Block (TCB).

Blocked Task
A blocked task is an active task that cannot execute. Blocked tasks do not compete for the CPU but do compete for memory at their current priority. A task typically enters a blocked state by waiting for an event flag associated with an event to set. Tasks can also be blocked by user commands.

BOO
BOO is the RSX system utility to bootstrap a named file as a system image. The current system context is completely erased. See Bootstrap.

Boot Block
The Boot Block is the first block on a disk. This block contains the code necessary to read a system image on the disk into memory and start execution of this image. The boot block has the starting physical block and length of the image to read. Boot blocks are created by the SAV program when the /WB switch is used. The PDP-11 ROM bootstrap loaders have only enough code to read this

block into memory and start its execution. The Boot block code then does the second phase of booting up a system. See also Bootstrap, INITL, and SAV.

Bootstrap
Bootstrap is the process of completely starting a PDP-11 with fresh copy of the RSX system. In RSX, this is a three step process. First, the hardware ROM bootstrap code reads the first block of the disk into memory. This block contains code to then read the entire system image into memory. Finally, control is passed to the system startup code, either INITL for a virgin system or SAV for a saved system.

BRO
BRO is the RSX system utility for outputting a message to specified terminals, either a single terminal, all logged in terminals, or all terminals. It is typically used to carry on a conversation between two terminals or to notify all users of some change in system status.

BRU
BRU is the main RSX system utility for making backups of disk volumes and restoring from the backup to a disk. BRU is highly optimized for speed and has support for various types of incremental backups and selective file restores.

Bucket
Bucket is the I/O and disk storage unit fore RMS relative and indexed files.

BYE
BYE is the RSX system utility for terminating a terminal session. BYE cleans up any current activity from the terminal by aborting tasks and marks the terminal logged out. Until a correct login sequence is completed, RSX will not accept any commands from the terminal after a BYE.

Cache
Cache refers to a technique of putting a high-speed, limited set of objects in front of a bulk storage of the objects. All accesses for an object are first checked to see if present in the cache, and if so, retrieved from it to speed access. The cache will have some algorithm for determining what objects to store and discard (look-ahead, least recently used, fixed).

On PDP-11's, cache is typically used for memory. High-speed memory is put in front of main memory on the PDP-11/44/60/70. RSX supports the use of cache on those processors. Caching is also sometimes used by operating systems for disk blocks, but not currently by RSX systems.

Catch-All
Catch-All is a MCR mechanism by which it passes any commands with verbs it does not recognize to a task named ...CA. The catch-all task can then take whatever action on the command line it pleases, including constructing new commands and passing back to MCR. See CCL.

CCL
CCL stands for Concise Command Language and is the most popular implementation of a catch-all task for RSX. CCL comes from the KMS Fusion kit on the SIG tapes. It is table driven and supports option prompting, command parsing, and MCR command submission.

CDA
CDA is the RSX system utility for outputting formatted, annotated listings of RSX crash dumps.

Checkpointing
Checkpointing is part of the process by which RSX makes memory space available to tasks. If a tasks is waiting to run and no memory is available for it, RSX will move any lower priority task from memory to disk to make memory space available for the task waiting to come into memory. Tasks which are moved out of memory will be brought back into memory when either a task exits and frees up memory space or they can checkpoint out an even lower priority task.

Checkpointing is invisible to the user and task and is an automatic process. The typical way you learn the checkpointing is occuring is because response time of your tasks is slower than normal.

CL:
CL: is a pseudo device that is always present in a RSX system. CL: stands for console listing device and is typically redirected to the console terminal. See Pseudo Device.

CLI
CLI stands for Command Line Interpreter. CLI's are a RSX system feature and provide the mechanism for a user to communicate via a terminal with the operating system. Whenever you sign onto a RSX system, a CLI is associated with your terminal. The CLI then gets all unsolicited input from the terminal and processes it according to the command language the CLI provides. The two CLI's

provided by Digital are MCR and DCL.

Clock Queue
The clock queue is a time-ordered list of future events. The queue is kept in the executive pool and includes mark time and task scheduling requests has well as internal RSX executive timed events.

Cluster Library
Cluster libraries are two or more resident libraries that share the same virtual task address space, i.e. the same APR. The overlay control code uses the PLAS directives to correctly map the appropriate cluster library when a routine is called.

CMP
CMP is the RSX utility to compare two source files and report the differences in a variety of ways.

CO:
CO: is the console output device. If console logging is not included in a system, CO: is a pseudo device typically directed to the console terminal. Otherwise, the CO: device is a real device used as a gateway to direct output to the console logging mechanism.

Command Dispatcher
The command dispatcher, MCR..., is a special RSX system task that takes all unsolicited input from terminals and passes to the appropriate CLI for that terminal. See Unsolicited Input.

Command Line Interpreter
See CLI.

Command String Interpreter
See CSI.

Connect-to-Interrupt
Connect-to-Interrupt is a feature of the RSX executive that allows a privilege program to connect its service code to a interrupt vector. This allows special devices to be serviced directly from a task without the need to write special device drivers.

Contiguous
Contiguous means a set of physical adjacent objects, typically refering to disk blocks. In a contiguous file, the entire block allocation is a set of physically adjacent blocks. This means only one operation is needed to map a virtual block to logical block in the file, simply adding the starting logical block number to the

virtual block number. Because contiguous files are much more efficient to handle, RSX requires certain files, specifically task images, to be contiguous.

Control Status Register
See CSR.

Co-Tree
One of one or more secondary tree structures within an overlay structure. All modules contained in a co-tree are accessible from any point in the main segment or a lower co-tree. Conceptually, a co-tree can be considered has a second, independent overlay structure within a task.

Common Block
Common block refers to either a Fortran common area or is another name for resident common.

Crash
A crash is a RSX system's response to an unstable condition. When the executive encounters a condition it is unable to handle, it stores certain volatile information on a special crash stack and enters special code to record all of memory to a scratch media. The system must then be bootstrapped to continue normal operations. See Crash Dump and CDA.

Crash Dump
Crash Dump is a copy of all of physical memory at the actual point the system encountered an exception it cannot handle. Special code in the executive has primitive routines for recording memory to various scratch media. This copy can be symbolically interpreted by CDA.

CRF
CRF is a slaved RSX task for processing cross-reference information accumulated by either the Macro assembler or task builder and appending a cross-reference listing to either the macro listing or task map.

Cross-Reference
Cross-Reference is information which shows all occurences of a symbol in either a macro source or task build. The Macro assembler and RSX task builder record each time a symbol is encountered. This information is passed to CRF for actual cross-reference processing.

CSI
CSI stands for Command String Interpreter and is a set of system library routines used to parse RSX command lines. Almost all RSX utilities use CSI for command

processing, so a consistent interface is provided across utilities.

CSR
CSR stands for Control Status Register. Each peripheral on a PDP-11 has a set of registers that allow control and operation of the device. These registers look like memory locations at certain fixed positions. The main register for a device is called the CSR by convention and is typically, but not always, the first register in the set. See I/O Page.

Dataset Descriptor
A dataset descriptor is a FCS control block that points to the device, UIC, and filename portions of a filename string.

DCB
DCB stands for Device Control Block. DCB's are the portion of the device databases which defines a class of like devices, such as all terminals, all RK05's, etc. The DCB names the device, has the mapping to the device driver, and specifies the legal functions serviced by the device driver.

DCL
DCL stands for Digital Command Language. DCL has several levels of definitions. DCL is a Digital standard for command languages that is supported across many operating systems. This provides a consistent user interface when moving from RSX to other systems.

DCL is also the RSX implementation of the standard. It is a task that is known to the system as a CLI. The command dispatcher, MCR..., passes unsolicited commands to DCL for parsing. DCL uses a table-driven mechanism to parse the commands and construct actual MCR commands to pass back to the command dispatcher for actual execution.

DECnet
DECnet is a layered product for RSX that allows network communications to other Digital operating systems.

Detach
Detach has two meanings in RSX. First, Detach is an I/O function (IO.DET) which releases a program's exclusive access to a non-shareable device. See Attach

Second, detach is a PLAS directive function that allows a task to release access to a region of memory.

Device Control Block
See DCB.

Device Driver
Device Driver is a RSX-11M/M-Plus term for the code which actual services a peripheral. Device drivers form the bottom of the RSX I/O mechanism and are perform the basic I/O functions for the device: read, write, special control. Device drivers need not reside inside the executive, but are mapped directly from the executive and execute in kernel mode.

Device Handler
Device Handler is a RSX-11D/IAS term for the code which actual services a peripheral. Device handlers are like device drivers, except that perform a part of their processing as a task and other service is kernel mode.

DIC
DIC stands for Directive Identification Code. Each RSX system directive has control block (DPB). The first word is the DIC, which consists of a low-byte directive identifier and high-byte DPB size.

Digital Command Language
See DCL.

Directive
Requests for system functions from user tasks are called directives. A user task forms a control structure that identifies the directive and holds parameters and issues an EMT instruction to pass control to the executive. RSX supplies a complete set of Macro-11 macros for assembly language programs to construct and issue directives. A similar subroutine library is provided for Fortran code.

The RSX directives allow tasks to obtain task and system information, measure time intervals, perform I/O, start and control other tasks, communicate with other tasks, manipulate the virtual and logical address space, wait for events in the system, and exit from the system. See DPB, DIC.

Directive Identification Code
See DIC.

Directive Parameter Block
See DPB.

Directive Status Word
See DSW.

Directory
A directory is a file that briefly catalogs a set of files stored on disk or tape. The directory includes the filename, type, and version and also has a pointer to the actual file.

RSX breaks directories into two classes. The MFD (Master File Directory) is the directory that catalogs the directory files. UFD's (User File Directories) are the actual catalogs for files.

Note that a file's directory entry is independent of the file itself and one can exist without the other. For example, temporary files are created without directory entries and therefore must be deleted when a program is finished with them.

Disk-Resident
Disk-resident is that which resides on a disk until needed.

Disk Swapping
Disk Swapping is a RSX mechanism for allowing tasks at the same priority to compete equally for memory. Disk swapping and its associated priority have nothing to do with the actual priority a task competes for the CPU, although obviously a task must be in memory in order to execute.

Disk swapping works on the basis of a swapping interval and priority. Each time a task is loaded into memory, the executive sets the priority it competes for memory to the task priority plus the swapping priority. At each swapping interval, all memory priorities are decremented. The minimum value is the task priority minus the swapping priority. After every interval, the executives checks to see if some task waiting for memory is now greater in priority than tasks currently in memory. If so, a checkpoint is initiated.

For an example, consider a partition large enough to hold either task A or B and swapping interval of 1 second and priority of 3. Both A and B have a normal priority of 50. When A is loaded, its memory priority is set to 53 and after one second drops to 52. When after 4 seconds the value drops to 49, the executive will checkpoint A out of the system and load in B. A will now have a priority of 50 and wait until task B's memory priority drops to 49.

DMA

DMA stands for Direct Memory Access. DMA is the technique fast devices like disks use to transfer to and from memory. The devices is told the starting physical address and size of transfer and then does the operation without any further program control. The device will then signal an interrupt when the transfer is complete.

DMO

DMO is the RSX system task for disabling access to a disk or magtape volume. This process is called dismounting.

DMP

DMP is a RSX utility for outputting RSX files in a variety of formats, including octal, ASCII, RAD50, and hexadecimal.

DPB

DPB stands for Directive Parameter Block. A DPB is the user task control structure that holds the information for the executive to use when processing a directive.

Driver

See Device Driver.

DSC

DSC is a RSX utility for making a volume copy of a disk to either another disk or magtape. DSC also performs the function of compressing the unallocated space on the disk into one contiguous fragment.

D-Space

D-Space refers to one of the two addressing spaces of the PDP-11 memory management unit. If enabled, all data references use separate D-space PAR/PDR registers to resolve the reference. The use of D-space allows the virtual address space of a task to expand to 64 KW's, 32 for instructions and 32 for data. Only RSX-11M-Plus supports D-space.

DSR

DSR stands for Dynamic Storage Region. See Pool.

DSW

DSW stands for Directive Status Word. This is a special word in a task which is used to return success/failure information for all directives. RSX convention is a positive number is directive success and a negative value is some from of directive failure.

Dynamic Storage Region

See Pool.

EAE

EAE stands for Extended Arithmatic Element and is an obsolete option used on early PDP-11's for multiple and divide support.

EDI

EDI is the traditional RSX editor. It is a line-oriented editor with a a very simple command set. It is the easiest and least complex of the editors available for RSX to learn.

EDT

EDT is the Digital standard editor and is available on most Digital operating systems. EDT main power comes from its video editting mode available for VT-series terminals.

EIS

EIS stands for Extended Instruction Set. EIS instructions are now standard features on PDP-11 processes and include the MUL, DIV, ASH, ASHC, XOR, and SOB instructions. EIS instructions are not available for early PDP-11 model (PDP-11/10 and 11/20) and is an option on some (PDP-11/35 11/40).

ELI

ELI is a RSX system utility for controlling the error logging system. See Error Logging.

EMT

EMT is a set of PDP-11 instructions that cause a trap to the executive through vector 30. EMT instructions are used in RSX to pass control to the executive for directive processing. Sometimes EMT is used as a synonym for RSX directives.

EOF

EOF stands for End-of-File and refers the the logical end point of a file. EOF's are a function of FCS/RMS and not Files-11. Both FCS/RMS maintain the last point data was written and do not allow read access beyond this point.

ERL

ERL is the RSX system task that actual performs error logging. The executive passes error packets to ERL which then records them to a disk file. See Error Logging.

Error Logging
Error logging is a RSX system feature to record various information about hardware errors and output annotated listings on the errors. The RSX error logging system handles errors detected by devices when performing I/O operations, interrupts that are unexpected, and memory and cache parity problems.

Event Flag
An event flag is a single bit indicator that is used to synchronize programs with events. A variety of RSX directives allow tasks to read, clear, and set event flags, as well as wait for an event flag to set. Also, almost all events in a RSX system, such as I/O completion, can be associated with an event flag. When the event occurs, the executive will set the event flag.

Event flags are addressed by number and range from 1 to 96. This range is further broken down into 3 32-bit classes: local, global, and group global.

EXCOM1/2
EXCOM1/2 are the names of the executive extension regions. These are partitions that holds executive code. The executive maps the regions directly when it needs them.

Executive
Executive is a general word for the code and processing which comprising the basic part of the RSX operating system.

Extend
Extend means to add additional space to the end of an object. In RSX this usually means to add more virtual address space to a task or more disk blocks to a file.

Extended Instruction Set
See EIS.

External Task Header
External Task Header's is a RSX-11M-Plus feature that puts the task header directly in front of the task image in memory and not in system pool like in RSX-11M.

F11ACP
F11ACP is the special RSX system task that implements the Files-11 disk structure. F11ACP is a ACP and process the Files-11 QIO's.

F4P
F4P is shorthand for the Fortran-IV-Plus compiler. This is a separate product and supports the ANSI 66 Fortran standard with extensions. F4P assumes the availability of the EIS and FPP instructions.

F77
F77 is shorthand for the Fortran-77 compiler. This is a separate product with origins from F4P. F77 supports a subset of the ANSI 77 Fortran standard.

FCB
FCB stands for File Control Block. FCB's are the executive data structure which hold the necessary information about open files. F11ACP creates a FCB for each time a file is first opened on the system. FCB's can come from either F11ACP's internal buffer space or system pool when that is exhausted.

FCP
FCP stands for File Control Primitives. These are the various QIO's which make up the functions available from a user program for accessing files on a disk or magtape volume.

FCS
FCS is shorthand for File Control Services. FCS is a set of routines linked directly to a task that provide device independent I/O and primitive record management. FCS subroutine calls break down the user I/O request into specific QIO's for the actual device.

FDB
FDB is an acronym for File Descriptor Block. This is the main control structure used by FCS and contains information about the type of I/O being performed from a user program.

File Control Block
See FCB.

File Control Primitives
See FCP.

File Control Services
See FCS.

File Descriptor Block
See FDB.

File Header

A file header is one disk block the holds all information about a file. It includes attributes used by FCS/RMS to determine the internal record structure and the mapping pointers for virtual disk blocks to the actual physical disk blocks.

File Name Block

See FNB.

File Specification

A file specification is the unique identification of a file. The convention for file specification is:

```
ddnn:[g,m]filename.typ;version
```

The device name (ddnn:) is a two-letter mnemonic, an octal number from 0 to 77, and a terminating colon. The UIC specification ([g,o]) is a pair of octal numbers that range from 0 to 377. UIC's are enclosed in brackets and the two numbers are separated by a comma. The filename is one to nine character alphanumeric string and the filetype is a one to three character string. They are separated by a period. The version number (;version) is an octal number which is always preceded by a semicolan.

File-ID

File-ID is a pair of numbers which uniquely identify each file on a volume. The first number is the position of the file header in the index file. The second number is called the sequence number and is the number of times the header has been used to for a file header. Whenever a file is created, the current sequence number is incremented by one.

The mapping from a file directory entry to the file header is by the File-ID. If a File-ID is already known for a file, directory processing can be skipped.

Files-11

Files-11 is a term for the on-disk structure used by RSX for files. It also applies to the various user functions available to access files.

FIS

FIS is a hardware option for the PDP-11/35 and 11/40 processors that provide four simple instructions for floating arithmetic.

Floating Instruction Set.

See FIS.

Floating Point Processor

See FPP.

FLX

FLX is a RSX system utility for transfering files between RSX systems and other PDP-11 operating systems. FLX understands RT-11 disk formats and DOS magtape formats.

FMT

FMT is a RSX system utility for formatting disk packs.

FNB

FNB is shorthand for Filename Block. This is the structure used by FCS to name files. The parsing logic of FCS takes dataset descriptors and template filename blocks to construct the actual FNB.

Fork

Fork is a term applied to the RSX executive's mechanism for serializing access to the executive and its databases. Whenever an interrupt service routine needs to drop processing down from interrupt level to executive level, it performs a fork. An example of this would be when a device error occurs and needs to be logged or the interrupt service routine detects I/O completion.

The fork process queues an entry to the fork list. When the executive finishes its current process, it checks the fork queue before returning to a user task.

FPP

FPP stands for Floating Point Processor and is a hardware feature for most PDP-11 CPU's to implement floating point instructions. FPP supports integer to floating conversion and single and double precision floating notation.

FTB

FTB is a RSX utility for fast task building. It has only limited powers but is three to four times faster for simple task builds.

Global Event Flags

Global event flags are event flag numbers 33 to 64. These event flags are seen by all tasks in the system and therefore can be used for intertask communication. For example, convention at a site could say the application system is to shutdown gracefully if global event flag 35 is set. All application tasks would then periodically check the state of this flag and being shutdown operations if they find it set. See Group Global Event Flags and Event Flags.

Global Symbol
: A global symbol is a symbol defined in one object module that can be referenced in another object module. Global symbols are identified and defined by the Task Builder. An example of a global symbols would be the subroutine name of a Fortran routine.

Group Global Event Flags
: Group global event flags are event flag numbers 65 to 96. There can actually be many sets of these event flags. The sets are addressed by a tasks group number. All tasks at the same group see the same set of group global event flags. There are executive directives to create and destroy group global event flag sets and to lock their existence.

Group Number
: Group number is the first part of the account (UIC) pair. Group numbers are expressed in octal and range from 1 to 377. See Account and UIC.

Handler
: See Device Handler.

Hang
: When a terminal, task, or system is going nowhere and doing nothing useful, it is said to be hanging. Hanging results from many different causes. Some of the more frequent are exhaustion of system resources, deadlock, and program loops.

Header
: See Task Header.

$HEADR
: $HEADR is a RSX system variable that has the address of the current task header. This location is updated whenever a context switch is made to another task.

HEL
: HEL is a RSX system utility to initiate a terminal session. HEL identifies you to the system, establishes any privileges you may have, and set the CLI to initially use from the terminal. HEL also processes all requests for HELP.

HOM
: HOM is a RSX system utility to change the volume attributes of a already initialized disk.

Home Block
: The home block is the root information for a Files-11 disk volume. The home block is usually the second physical block on a disk. If this block is bad, the home block is placed at the first good block found at multiples of 256. The home block has all necessary information for F11ACP to find the rest of the Files-11 structures.

ICP
: ICP is the RSX system task for processing indirect command files.

Index File
: Index File is the area on a disk structure which comprises the control information for the Files-11 volume. The index file shows up as an actual file on the disk ([0,0]INDEXF.SYS). It contains the boot and home blocks as its first two blocks, followed by other special information. The majority of the index file is made up of individual file headers.

Indirect Command File
: An indirect command file is a file processed by the ICP task. The command file consists of special directives that control the processing done by ICP and commands for ICP to pass to the CLI. The directives available in indirect command files are sufficient to consider them as a programming language: flow control, input and output, and variable and character operations.

IND
: IND is the RSX system task for processing indirect command files available on RSX-11M V3.2 and earlier systems. IND was enhanced and renamed to ICP in RSX-11M V4.0.

INI
: INI is the RSX system utility used to create the initial Files-11 on-disk structure. INI creates a blank index file and establishes the volume characteristics. Doing an INI on a volume completely loses any previous structure on the disk.

INS
: INS is the RSX system utility for installing a task. See Install.

Install

Install is a term in RSX that refers to making a program known to the system and available for execution. All programs are contiguous files known as task images. Until a task image is installed, it cannot be executed. Installing a task is a process of reading information from a task header and constructing a TCB. See TCB.

Interrupt

The mechanism used by PDP-11's to signal changes in device status is known as an interrupt. There are also instructions (EMT, TRAP, BPT, IOT) and illegal conditions (odd address, illegal instruction) that are also interrupts.

When an interrupt occurs, the current program counter and processor status are pushed onto the stack and a interrupt vector is used to supply a new program counter and processor status. This causes control to pass to code to service whatever specific condition is being signaled.

Interrupt Vector

An interrupt vector is an transfer vector, processor status pair in the first 1000 words of physical memory. Each interrupt in a PDP-11 are tied to a specific interrupt vector so control can be passed to the correct service code.

Interrupt Transfer Block

See ITB.

I/O Mechanism

I/O mechanism is a term applied to the various components used in a RSX system for device input and output. The I/O mechanism can be broken down to four distinct levels: FCS/RMS, QIO, ACP's, and device drivers.

FCS/RMS comprise the top level of the I/O mechanism and provide for device independence and record management. These are routines linked directly to user tasks and perform much of the bookwork required to interface with the lower levels.

The QIO is the directive used to signal I/O from a user task to the RSX executive. All I/O, including that actually performed by FCS/RMS is done using a QIO. The executive performs common processing on the request and then forwards it to the specific function handler. This can either be an ACP or device driver.

ACP's are tasks which implement high-level protocols for classes of devices, such as the on-disk structure or ANSI labeled tapes. Many of the QIO's issued from user tasks are not specific to the device but instead routed by the executive to ACP's. These tasks process the request in a variety of forms, which include issuing QIO's of their own for specific device functions.

Device drivers form the bottom of the I/O mechanism and perform device specific I/O operations.

I/O Packet

An I/O packet is the internal executive representation of a QIO directive. The I/O packet is the control information passed to either an ACP or device driver for further processing.

I/O Page

All PDP-11's consider the last 4 KW's of their physical address space as a special area known as the I/O page. Instead of memory, this space has device registers for all peripherals on the system.

I/O Rundown

I/O rundown is a term for the state a task enters when it tries to exit but all outstanding I/O cannot be completed. RSX keeps track of outstanding I/O requests and open files for a task and tries to clean these up when a task exits. If an error in the system causes the cleanup to fail, the task will get stuck in I/O rundown state and remain in limbo until some other action is taken (like rebooting the system).

IOX

IOX is a RSX system utility for exersing the devices on the system. IOX will place a heavy I/O load on as many parts of the system as the user selects.

I-Space

I-Space refers to one of the two addressing spaces of the PDP-11 memory management unit. If enabled, all instruction references use separate I-space PAR/PDR registers to resolve the reference. I-space is also the default mode for all references for memory management units that do not have D-space implemented and for those units that have D-space disabled.

ITB

ITB is shorthand for Interrupt Transfer Block. This is an executive data structure used to pass control for an interrupt to a loadable device driver. The interrupt vector for the device points to the ITB and the ITB then maps the loadable device driver and passes control to its interrupt service routine.

KMS Fusion Kit
: The KMS Fusion Kit is a major submission on the RSX SIG tapes that covers two primary areas: RSX-11M accounting and CCL. The kit has been submitted to many tapes and is always found in [344,*]. The latest version for RSX-11M V4.0 is on the Atlanta Spring 1982 tape.

Kernel
: Kernel is a term used in RSX to refer to any type of executive processing and in general a synonym for executive.

Kernel Mode
: Kernel mode is a processor state on the PDP-11. It is the most privilege of the three processor states (kernel, supervisor, user) and has no restrictions on instructions that can be executed. All executive processing is done in RSX in kernel mode. Switches to kernel mode from higher levels are accomplished through the interrupt vectors, which set the vector address and processor state. When in kernel mode, the memory management unit uses the kernel address registers for virtual to physical memory mapping.

Kernel Stack
: The kernel stack is the stack used by the executive when processing in kernel mode. It sits immediately above the vector space. A PDP-11 that supports memory management has a separate stack pointer register for each processor mode. When a switch is made from one mode to another, the stack pointer register is also switched.

Layered Product
: A RSX layered product is a separate product available from Digital that requires RSX. Examples of layered products include all languages (F77, Basic-Plus-2), DECnet, and SORT-11.

LB:
: LB: is a pseudo device that is redirected when a system is bootstrap to the disk booted from. LB: is the default disk for almost all RSX system files.

LBR
: LBR is a RSX utility for creating and manipulating library files. See Library.

LCB
: LCB is shorthand for Logical Control Block. An LCB is an executive data structure and holds the information necessary to translate a logical device assignment to the real device. The MCR ASN command is used to create, destroy, and list LCB's.

LDR
: LDR is a RSX-11M system task that is directly interfaced in the executive. LDR is used to load task images and for the checkpointing of tasks to and from disk.

Library
: A library is a file which contains related types of files. The primary use of library files is to hold collections of macro definitions and object modules. The Macro assembler and task builder have facilities to retrieve macros and objects from these libraries. RSX also provides a type of library termed universal. Such libraries can hold any type of file and are a convenient method for packaging many small files.

LOA
: LOA is a RSX system task that is used to read a loadable device driver into memory and initialize the system so the device can then be used for I/O operations. See Loadable Driver.

Loadable Driver
: A loadable driver is a device driver that is not initially built into the kernel executive. Instead, the driver is loaded into a partition using LOA or VMR. The advantage of loadable drivers is that the driver does not take up space in the kernel executive. This frees space for pool. See Device Driver.

Loadable Database
: Loadable database refers to a loadable device driver that follows some specific conventions so the actual device database is a part of the driver image on disk. When the loadable driver is loaded into the system using LOA or VMR, the template database in the image is used to created the normal device database. Space for this database comes from system pool. The advantage of loadable databases is that they allow new devices to be added to the system without an entire new system generation. Note, loadable databases are not destroyed when the driver is unloaded.

Loader
: See LDR

Local Event Flags
Local event flags are event flag numbers 1 to 32. Each task has its own copy of local event flags.

Local Symbol
A local symbol is a definition that cannot be referenced outside of the object module that has its definition.

Local User Group
See LUG.

Locked Block
A locked block is a block is a disk file that can only be read and written by the task that locked it. This mechanism is used in RSX-11M to allow multiple tasks to open the same file and still safely allow update access to the file.

Logical
Logical is used in RSX as a synonym for physical. See Logical Block, Logical Address Space, and Virtual.

Logical Address Space
The total amount of physical memory to which a task has access rights. The memory management unit performs the translation between virtual addresses and logical addresses. The PLAS directives allow a task to change their virtual address to map different logical addresses. See Virtual Address Space.

Logical Block
A logical block is the actual block on the disk. F11ACP and the executive are responsible for mapping a file virtual block to the actual logical blocks. The Window Block has this mapping information. See Virtual Block.

Logical Control Block
See LCB.

Logical Device
A logical device is a device name that maps into a physical device. Logical devices are created, listed, and destroyed by the ASN command and provide a useful method for assigning devices independently of the actual devices used at any one point in time.

Logical Unit Number
See LUN.

Lost File
A lost file is a file that has no directory entry and therefore can only be accessed by file-ID. One function

of the VFY utility is to find lost files and create a file directory entry for them.

LUG
LUG is an acronym for Local User Group. LUG's are small groups of DECUS formed in a specific area by users with specific interest.

LUN
LUN stands for Logical Unit Number. A LUN is a number associated with a physical device during a task's I/O operations. Each task establishes its own correspondence between LUN's and devices.

MAC
MAC is the name of the RSX assembler. The assembler language is called Macro-11 and has an extensive set of directives in addition to support the actual instruction notation.

Macro
A macro is a single assembly-language statement that expands into a predefined set of other assembly-language statements. Macros may have arguments which are substituted by the MACRO-11 assembler when expanding the macro.

Macro Library
A macro library is a collection of macros packaged into a file by LBR. The Macro-11 assembler has a directive (.MCALL) which will cause searches of macro libraries.

MAG
MAG is a RSX system utility for manipulating magnetic tapes.

Manual Load
This is a method of loading overlay segments in which the user makes explicit calls in his task to load overlays and handle unsuccessful load requests.

Mapped
Mapped refers to a RSX system that has support for the PDP-11 memory management unit.

Mapping directives.
A synonym for the PLAS feature of RSX. Refers to the set of directives which allow a task to change its virtual addressing.

Mark Time
The Mark Time directive cases an event to occur at some interval in the future. The task can be notified when the time elapses by either a event flag or AST.

MASSBUS
MASSBUS is a name for the disk and magtape controllers used on a PDP-11/70. These controllers provide direct access to 11/70 memory.

Master File Directory
See MFD.

MCR
MCR stands for Monitor Console Routine, the prime interface for a user with a RSX system. MCR receives all commands and either operates on them diretly or dispatches them according to its special parsing rules. MCR commands tend to use initials or acronyms in a strict syntax, rather than the English like syntax of DCL.

Memory Management
Memory management refers to the hardware option on most PDP-11's that allows the system to use more than the 32KW's directly addressable in the PDP-11 16-bit address. The memory management system takes each 16-bit address and maps it to the correct physical address.

Memory-Resident
In general, that which resides in memory all the time. The entity, as in the case of memory-resident overlays, may initially reside on disk. Once loaded it stays in memory.

Memory-Resident Overlay
An overlay that shares virtual address space with other overlay segments, but which resides in its own physical memory. The segment is loaded from disk only the first time it is referenced. Thereafter mapping directives are issued in place of disk load requests.

MFD
Each Files-11 volume has a Master File Directory (MFD). This is a special file that contains pointers to all User File Directories on the system. The MFD itself is directory [0,0].

Monitor Console Routine
See MCR.

MOU
MOU is the RSX system utility for reading the Files-11 information found on a disk and setting up the necessary RSX data structures so the files on the volume can be accessed.

MTAACP
AN ACP task that implement support for ANSI labeled tapes. MTAACP uses Files-11 QIO's to access such tapes.

Multi-Tasker
The Multi-Tasker is the newsletter of the RSX-11/IAS SIG. It has been published monthly since 1976, except for the current year where budget cuts forced temporary bimonthly schedules.

Multi-User Task
An RSX-11M-Plus or IAS task that has the read-only region of the code shared among several copies of the same task. Each task has its own copy of the read/write data areas.

NL:
NL: is the null device, a special device driver that provides a sink for all output and EOF signals for all input. The devices main use is to discard listing output. This is done by assigning a logical device to NL: whenever listings are not desired.

Null Device
See NL:

Object Library
An object library is a collection of object modules packaged into a file by LBR. The task builder understands the format of object libraries and get extract modules from them to resolve global references.

Object Module
An object module is a file that is the output of the Macro-11 assembler or the various RSX language compilers. Object modules have binary representation of the code desired by the programmer and are used as input to the task builder for creating the executable image.

Object Time System
See OTS.

OCB
OCB stands for Offspring Control Block. This is the executive data structure used by RSX to tie offspring task's to their parent tasks. OCB's are created in sys-

tem pool and linked to the offspring task.

ODL
ODL stands for Overlay Descriptor Language and is the type of file supplied to the task builder to define how it should setup overlays.

ODT
ODT is the On-line Debugging Tool. ODT is an object module that when linked with a task allows the user to examine and deposit locations in the task, set breakpoints, and single step the task's execution.

Offspring Control Block
See OCB.

OTS
OTS is shorthand for Object Time System. Each language available on RSX has a set of object modules that are used by the code generated by the language compiler. This code, called OTS, is linked to user programs automatically by the task builder. Because all programs written in the same language will use the same OTS routines, a common practice is to link the OTS routines in a resident library and share the code among tasks.

Overlapped Seek
Overlapped seek refers to starting the search for a specified disk block on one disk drive while performing a disk transfer on another drive. Most of the disks supplied by Digital have this ability but only RSX-11M-Plus implements this feature.

Overlay Description Language
See ODL.

Overlay Runtime System
A set of system library routines that the task build automatically links into an overlaid program to perform all overlay load and mapping requests.

Overlay Segment
A segment that shares virtual address space with other segments and is loaded when needed.

Owner Number
Owner number is the second part of the account (UIC) pair. Owner numbers are expressed in octal and range from 1 to 377. See Account and UIC.

Page Address Register
See PAR.

Page Descriptor Register
See PDR.

PAR
PAR stands for Page Address Register. It is also refered to as APR or Address Page Register. PAR's are a part of the PDP-11 memory management mechanism and there are a total of eight PAR registers for each of the various addressing spaces (kernel, user, etc.) A PAR register contains the starting physical address of a 4 KW virtual address.

Partition
A partition is a contiguous area in memory in which tasks are loaded and executed and data can be stored. RSX has three types of partitions: tasks, common, and system. The first two are static and can be used for tasks and data respectively. System partitions are allocated dynamically into subpartitions.

Partition Control Block
See PCB.

PAT
PAT is a RSX utility for making patches to object modules. PAT is used by Digital to correct distribution software which is not in source form.

PC
PC stands for Program Counter and is register 7. This is a special register and always contains the next address to fetch for instructions.

PCB
PCB stands for Partition Control Block. PCB's are the executive data structure used to define each region of physical memory that is in use. PCB's are linked together in pool by increasing physical address. The space between two successive PCB's is is the unused space that can be allocated.

PDP
PDP stands for Programmable Data Processor. When Digital first made computers, the industry would not accept a computer for less than $100,000. Digital decided to name it's machines Programmable Data Processors to get around the prejudice of the time.

PDR
PDR stands for Page Descriptor Register. PDR's are used by the memory management unit to describe the length of a page and the type of memory access allowed.

Physical Address
The actual byte locations in real memory.

Physical Device
A physical device is a real device. Logical and pseudo devices will finally map into a physical device.

PIC
An acronym for Position-Independent Code. PIC allows the code to be placed anywhere in a task's virtual address space without relocating the actual addresses.

PIP
PIP is the RSX utility for handling files. PIP major functions are copying files, listing directories, and deleting files.

PLAS
PLAS stands for Program Logical Address Space and is a set of directives that allow a task to create and destroy regions in memory and change its virtual mapping in these regions. PLAS allows a task, under program control, to overcome the 32KW addressing restriction of the PDP-11 by providing a user interface to the memory management unit.

PMD
PMD is a RSX system task for output a formatted dump of a task or particular part of a task (call snapshot). PMD is normally used to dump a task when it aborts so you can find the reasone for the error.

PMT
PMT is the special RSX system task used for pool monitoring. See Pool Monitoring.

Pool
Pool refers to the space in the executive used to hold RSX executive data structures. The structure of RSX-11M limits the size of the kernel executive to the first 20KW of physical memory. Pool is whatever space is left in this space after subtracting out the kernel code.

RSX-11M-Plus extends the available pool because the executive uses I/D-space and a second area for some structures.

Almost every RSX function requires some allocation of pool. If the space cannot be found, and error is returned and the operation fails. Low pool is the most common type of failure for a RSX system.

Pool Monitoring
Pool monitoring is a RSX executive feature that allows some action to be taken when system pool is critically low. The RSX executive keeps track of pool allocations and will notify the pool monitoring task (PMT) when various threshholds are crossed. If pool becomes exhausted, the pool monitor task will take complete control of the system and allow the system manager to abort selected task to recover their allocations.

Pool Fragmentation
Pool fragmentation refers to the state when enough pool is free for most requirements but the space is so scattered into little chuncks that allocations cannot be made. Pool allocation is by a first-fit algorithm and over time the free pool will become fragmented.

Pop
Pop means removing an item from the stack.

Position Independent Code
See PIC.

Post-Mortem Dump
See PMD.

Privilege Task
A task that has no restrictions on the executive services it can use. A class of privilege tasks are mapped to the executive and the I/O page and therefore can directly address pool structures and devices.

Processor Status Word
See PSW.

Program Counter
See PC.

Program Logical Address Space
See PLAS.

Protection Code
Each file has a protection code that specifies what access different categories of users have to the file. The catagories are selected by comparing the UIC associated with the file and the UIC of the program making the access. RSX has four categories: system, owner, group,

and world. For each category, four levels of file protection can be named: read, write, extend, and delete access.

PSECT
Memory allocation for various code segments when a program is linked in established by attributes assigned to program sections. PSECT is the Macro-11 directive for setting these attributes.

Pseudo Device
A pseudo device is an entity treated as an I/O device by the system by is not a real device. Instead pseudo devices are always mapped to an actual physical device. Pseudo devices allows programs to refer to devices without actual knowing the real name. For example, LB: always points to the system disk no manner which actual disk is booted.

PSW
PSW is shorthand for Processor Status Word. The PSW is the last word in the I/O page (and therefore in physical memory space) and has the current execution status. The PSW is broken into three bits fields. The first has the current and previous processor state (kernel, supervisor, and user) and determines what memory management registers will be used. The next is the processor priority and ranges from 0 to 7. This controls what devices will be allows to interrupt the processor. The final bits are condition codes and have information on the result of the last instruction.

Push
Push means adding an item to a stack.

QIO
QIO is the directive RSX uses for all I/O requests.

QMG
QMG is the RSX task that manages the queue of jobs directed to the batch processor, line printer, or other spooled devices.

Queue
Q queue is a waiting line of items waiting to be processed. RSX has a queue manager for batch and spooled jobs. Also, almost all executive lists are maintained as queues ordered by task priority.

Queue Manager
See QGR.

RAD50
RAD50 is a coding convention for the uppercase alphabetic characters, decimal digits, blank, period, and dollar sign. Three of these characters can be packed into one word. The executive uses RAD50 packing for almost all of its character storage as a space saving tool.

Record Management Service
See RMS.

Reentrant Code
Reentrant code means code that can be executed by more than one process at a time. Reentrant code must keep all local variables on a stack.

Region
A region is a contiguous block of physical memory in which a task, driver, resident common, or library resides. The PLAS directives allows tasks to dynamically create regions.

Remove
Remove means to eliminate the TCB for a task so it can no longer be executed.

Resident Common
A resident common is a partition that holds data. Resident common's can be mapped by many tasks at the same time. This allows data to be shared and exchanged between tasks.

Resident Library
A resident library is a partition that holds instructions. Resident libraries can be mapped by many tasks at the same time. This allows reentrant code to be shared among many programs, reducing the physical memory needed to run the programs.

RMD
RMD (also known as RMDEMO) is the RSX system utility for dynamically displaying the system status on a CRT screen. RMD will update its display once a second so you can visually see tasks enter and leave memory.

RMS
RMS stands for Record Management Services and is the more sophisticated of the two sets (FCS/RMS) of I/O routines available on RSX systems. RMS has routines for normal sequential I/O and more complex record management of relative and index (keyed) files. RMS is common to many other Digital operating systems so programs written using RMS I/O are usually more portable that those writ-

ten with FCS.

RNO
See Runoff.

Round-Robin
Round-robin is a RSX system feature that gives tasks at equal priority in memory equal access to the CPU. Without round-robin scheduling, the executive would give the first task at a priority the CPU and not interrupt it for other tasks at the same priority. Round-robin works by rotating tasks at the same priority in the ATL at some fixed interval and causing a significant event. Note, round-robin scheduling has no effect on memory allocation. See Disk Swapping.

RPT
RPT is the RSX system utility for producing error log reports. RPT reads the binary information accumulated by the error logging system and outputs annotated information on each error.

Runoff
Runoff is a DECUS utility for document preparation. Runoff takes a mixture of free-formatted text and formatting commands to output justified, paginanted output. This document is prepared using Runoff.

SAV
SAV is the RSX system task that takes a running, quiet RSX system and copies it the the system boot file. It then restarts the system from this file.

Save
Save in RSX refers to setting up a bootable system using the special system task SAV.

SCB
SCB is an acronym for Status Control Block and is the RSX executive structure that has the information about each device controller in the system.

Secondary Pool
Secondary pool is a RSX-11M-Plus feature where certain executive structures are allocated from a second pool located in the system partition. This pool can be any size and helps increase the available space in system pool.

Seek Optimization
Seek optimization refers to a technique of examining the waiting I/O requests to a disk and selecting the next

request based on some algorithm that reduces disk I/O time. This functionality is only available on RSX-11M-Plus.

Send-by-Reference
Send-by-Reference is a set of directives that allow two tasks to exchange information on a region. These directives area part of the PLAS functions.

Send/Receive
Send/receive refers the mechanism of one task sending a fixed-length (13 word) message to another task.

SHF
SHF is a special RSX system task that moves tasks in a system partition to make more contiguous free space available for loading tasks from disks. Whenever RSX tries to load a task in memory and fails, it starts the shuffler which attempts to consolidate the partitions and free space at the top of the main partition.

Shuffler
See SHF.

SIG
SIG stands for Special Interest Group. SIG's a major part of the DECUS structure and subdivide DECUS into members with similar interest. DECUS currently has 23 active SIG's. Symposia sessions are oriented by SIG's and each SIG publishes a newsletter.

SIG Tapes
At each symposia since 1977, the RSX SIG has collected software from members and put on one master tape. This tape is then copied and distributed via LUG's to all interested members. This collection is called the SIG tapes. Since the success of the RSX tapes, most other SIG's have initiated similar efforts.

Significant Event
A significant event is declared whenever there is a change in RSX system status, i.e. when an event occurs. When significant events are declared, RSX will review the status of all blockes tasks to see if the event has unblocked them and made them eligible to run. Note, setting an event flag is often associated with a significant event but is not a significant event itself.

Slaved Terminal
A slaved terminal is a terminal that has no access to CLI's. Therefore no program can be initiated from a slaved terminal using MCR/DCL.

SLP
SLP is the RSX utility used to make corrections to source files. SLP is a line-oriented editor that accepts text files with edit changes.

Software Performance Report
See SPR.

SP
SP stands for Stack Pointer and is a special PDP-11 register (R6). The various transfer control instructions use this register to save and restore the calling status.

Special Interest Group
See SIG.

Specified AST
A specified AST is an AST a task presets so it can receive AST notification if the event occurs in the future. Examples of specified AST's include one for when a message is sent to a task or the system recovers from powerfailure. There is a separate directive for each type of specified AST.

SPM-11M
SPM-11M is a layered product for RSX-11M that collects RSX performance data and summarizes this data for analysis. SPM puts hooks into the executive and records the occurence of almost every system event.

Spooling
Spooling refers the the process of sending I/O to a disk file and then queuing for transfer to sequential, non-shareable devices like line printers.

SPR
SPR stands for Software Performance Report and is the form used by Digital for users to report errors in software.

SRD
SRD is a popular DECUS utility that has a wide variety of directory operations. The major function is to sort directories and select files based on a wide variety of criteria.

SST
SST is an acronym for Synchronous System Trap. A system trap is a transfer of control that is caused by some event. RSX defines two types of system traps, synchronous (SST) and asynchronous (AST).

An SST is a trap that occurs synchronous to program execution. That is, the trap occurs as a direct result of program execution and will always occur in the same place if the same instructions are executed. SST's usually indicate a program error but can also be caused by specific instructions. Examples of SST's include odd address errors, TRAP instructions, and breakpoint traps (BPT instruction).

A program can specify SST notification for different types of SST's. When the trap actually occurs, the executive interrupts the current execution and passes control to the SST service routine. If no SST routine is enabled, the executive will abort the task.

Stack
A stack is a last-in, first-out temporary storage area. The instruction set on the PDP-11 has certain optimizations that make stack operations easy.

Stack Depth
Stack Depth refers the mechanism the executive uses to determine what is the current execution state and what kind of transition just took place. A system variable, $STKDP, is decremented each time there is an interrupt and incremented each time a return from interrupt is made. The variable is set to one if running in a user task, zero if at executive state, and negative if processing device interrupts.

Stack Pointer
See SP.

STB
STB is an acronym for symbol table and is a form of file output by the task builder. STB files hold the definition for all global symbols in the task image.

STD
STD stands for System Task Directory and is a linked list of all TCB's. Therefore, the STD is the list of all installed tasks in the system.

$STKDP
$STKDP is the global RSX system variable used as the stack depth counter. See Stack Depth.

Stop State
Stop state is a special wait state for tasks. A task in stop state does not compete for either the CPU or memory. It can be swapped out by any task, including those of lower priority. Otherwise, a task in wait state does

not compete for the CPU but does retain its priority as far as memory allocation is concerned.

Subpartition
A subpartition is a subdivision of a system partition that is made by RSX to hold tasks and regions created by the PLAS directives.

Supervisor Mode
This is one of the processor modes of a PDP-11 and has a separate set of memory management registers and stack pointer. Supervisor mode support is only available with RSX-11M-Plus for supervisor libraries. This allows resident libraries to be used without subtracting from the task's user mode address space.

Swapping
See Disk Swapping.

SY:
SY: is a pseudo device that stands for the users default disk device. Each user has a unique SY: assignment. This allows easy distribution of users across various disks.

Synchronous System Trap
See SST.

SYSCM
SYSCM is the executive module that holds all RSX global variables. These include the listheads for all executive data structures, current time information, and other executive control fields.

Sysgen
Sysgen refers to the process of creating a unique RSX system for your specific site needs. The process involves selecting the specific features and devices, assembling the executive source modules for those features, linking the executive and system tasks, and setting up the system as the bootable RSX system.

SYSTB
SYSTB is the executive module that holds the device data bases. This file is actually created on the fly by the sysgen process so it is unique to your system.

System Control Block
See SCB.

System Controlled Partition
A system controller partition is a type of partition that the RSX system controls allocation within. RSX will automatically create subpartitions in a system partition whenever a task load is requested.

System Library
The system library is an object library that TKB defaults to to resolve and undefined symbols after searching all other named libraries. The system library is named LB:[1,1]SYSLIB.OLB and typically holds the FCS routines, overlay system, and default language OTS modules.

System State
System state means executing in the executive mode.

System Task Directory
See STD.

Task
The task is the fundamental, executable RSX program unit. The tradition computer program in a RSX system is a task. Tasks are created by the task builder from object modules and libraries.

Task Builder
See TKB.

Task Control Block
See TCB.

Task Header
A Task Header is the low virtual address of a task that hold the task's initial and current context.

Task Loader
See LDR.

TCB
TCB stands for Task Control Block and is the RSX executive structure that holds all the information necessary to run a task. TCB's are created when the task is installed and destroyed when it is removed. The STD is the list of all TCB's. The TCB's in this chaine that are for active tasks are also chained together in a separate list called the ATL.

TECO
TECO is a character-oriented text editor available through DECUS and the RSX SIG tapes. TECO is the one editor available on all Digital operating systems. Its

command set is so powerful that TECO could properly be considered as a programming language.

Telephone Support Center
See TSC.

TI:
TI: is a special RSX device that always references the terminal from where a task was initiated. A program can do I/O to the user terminal by assigning a LUN to device TI: and the executive will map to actual terminal. Normally LUN 5 defaults to TI:.

TKB
TKB is the RSX task builder. Essentially, TKB takes object modules as input and relocates the code to the correct virtual address and resolves global references. TKB outputs a task image. This is a file which is ready to be installed and executed.

TKN
TKN is a special RSX system task that performs the error message processing when a task terminates abnormally.

$TKTCB
$TKTCB is a RSX system variable that always contains the address of the TCB for the current, active task.

TPARS
TPARS is a finite-state parsing mechanism that can be used to process any grammar. TPARS is implemented by a set of macros the user uses to describe the grammar and an object module that parses the supplied grammar according the states setup by the macros.

Transparent Spooling
Transparent spooling refers to capturing all output to a spooled device (line printer) and writing it to disk. When the I/O is complete, the mechanism queues the output for printing to the device. RSX-11M does not have transparent spooling. IAS and RSX-11M-Plus have this mechanism.

TSC
TSC stands for Telephone Support Service. TSC is a 24 hour, seven day a week service for answering questions about RSX.

UCB
UCB is an acronym for Unit Control Block and is the executive data structure for each device unit.

UFD
UFD stands for User File Directory and is a file that contains the names of all files found in the user account. UFD is also the RSX system task that creates the initial empty directory files.

UIC
UIC is the two-number user identification code that RSX uses to identify different people. UIC's are expressed in the form [g,o] with "g" being the group number and "o" the owner number. People working together are typically put in the same group because the file protection mechanism allows group members to have access to each others files.

UMR
UMR stands for Unibus Mapping Register and is the hardware feature used on 22-bit PDP-11's (PDP-11/70, PDP-11/44) to map the 18-bit Unibus to the 22-bit memory.

Unibus
Unibus is the common I/O bus structure used on a PDP-11 to connect devices to the CPU.

Unibus Mapping Register
See UMR.

Unit Control Block
See UCB.

Universal Library
A universal library is a library file that holds a common set of files. The librarian utility, LBR, is used to create universal libraries and extract and list the individual contents.

Unsolicited Input
Unsolicited input is any terminal input that occurs when no program has an outstanding read to the terminal or has the terminal attached or slaved.

User File Directory
See UFD.

User Identification Code
See UIC.

User Mode
User mode refers to execution of user programs as opposed to execution with the RSX executive.

VCB
VCB stands for Volume Control Block. VCB's are the executive data structure used by ACP's to hold information unique to each mounted volume.

VFY
VFY is the RSX utility for checking the integrity of a Files-11 disk volume. VFY will also correct many of the common errors that can occur with a Files-11 disk.

Virgin System
A virgin system is a RSX executive task image file that has never been booted or modified using VMR. The recommended practice is to copy the virgin system to another file before setting up with VMR and booting.

Virtual
The term virtual means from the point-of-view of the program as opposed to logical which means from the point-of-view of the system or hardware.

Virtual Address
Virtual address is the address within a task and from the task's point of view. Virtual addresses are limited to 32 KW's (0-177777) and can be dynamically changed using PLAS directives to different logical addresses.

Virtual Block
Virtual blocks are blocks in a file. These are numbered starting from one.

Virtual Disk
Virtual disk is a program on the RSX SIG tapes that allows contiguous files to be treated as disks.

Virtual Terminal
Virtual terminal means a device that functions as a terminal but is not associated with hardware. Instead a program acts as the terminal keyboard and screen.

VMR
VMR is the RSX utility for setting up the initial state of an RSX system when it is booted. VMR commands are identical to MCR, except where MCR operates on the data structures in pool, VMR operates on a disk file.

Volume
Volume is the largest logical unit of the file structure and is equivalent to a disk pack. Volumes are then broken down into files.

Volume Control Block
See VCB.

Wait State
Wait state refers to a task which is active but not competing for the CPU. The task is waiting on some event or has its execution suspended.

Wildcard
Wildcard is a special character in a file specification that matches anything for the field. RSX has two forms of wildcards: asterisk (*) for matching all characters in the position and percent sign (%) for matching exactly one character.

Window
A window is a task's view of a region of memory. Windows are controlled by the PLAS directives and can be moved under program control to any point in a task's regions.

Window Block
Window block is the RSX executive structure used to map virtual blocks in a file to the actual logical (physical) disk blocks.

XDT
XDT is a part of the RSX executive for debugging of executive code. XDT uses a subset of ODT commands. From the system console, a programmer can set breakpoints in the executive, examine and modify locations, and step the execution of the executive. When in XDT mode, the executive does not process any normal activity.

ZAP
ZAP is a RSX utility with ODT-like commands for examination and modification of disk files. ZAP has special features to easy use with task image files.

BRU Sorting Bug - A Wrapup

Carl T. Mickelson

Goodyear Aerospace Corp
D470/G3
1210 Massillon Rd.
Akron, Ohio 44315

In an SPR submitted to DEC in May, and published in the Multi-Tasker in July 82, I outlined a bug-fix to the LBNORD module of BRU. The last issue of the Multi-Tasker contained a followup article in which I presented a correction to LBNORD to stack the larger partition of the sort list rather than only the left half as originally distributed by DEC.

This article is written to correct an error in the patches that appeared in that article. Following are the corrected patches - note that only one instruction has changed from that published earlier. Also, the checksum for the LBNORD.POB patch object module is changed by the corrected instruction.

```
   PAT
LBNORD.OBJ;2=LBNORD.OBJ;1/CS:10755,LBNORD.POB;1/CS:33041
```

The patch for V4.0 given here is slightly different in patch locations than the V3.2 patch due to other changes DEC made in the baseline version for V4.0. Since we are still running V3.2, I have not been able to try this patch on a V4.0 BRU.

```
  .TITLE  LBNURD          ; FOR V3.2 BRU
 ;
 ; MODIFICATION:
 ;
 ;         01.2 -- FIX SORT LOGIC ERROR IN LBNORD (21 MAY
1982)
 ;                 THIS IS PART OF THE REAL FIX FOR
5.1.17.9
 ;                 C. T. MICKELSON, GOODYEAR AEROSPACE
CORP.
 ;                 AKRON, OHIO 44315 (216) 796 - 2388
 ;
 ;         01.3 -- FINISH CORRECTION TO QUICKSORT BY
STACKING
 ;                 LARGER PARTITION OF LIST RATHER THAN
ALWAYS
 ;                 THE LEFT PARTITION THIS ELIMINATES
 ;                 POSSIBILITY OF A STACK OVERFLOW ERROR
BUT MAY
 ;               MAKE PAST BRU TAPES INCOMPATIBLE.
 ;                 C. T. MICKELSON, GOODYEAR AEROSPACE
CORP.
 ;                 AKRON, OHIO 44315 (216) 796 - 2388
 ;
  .PSECT
  .BLK.=.
```

```
 .IDENT  /01.2/
 .=.BLK.+436
 DEC     $QSTAK+200.(R5)

 .IDENT  /01.3/
 .=.BLK.+12
I:       .BLKW   1
J:       .BLKW   1
ISTACK:  .BLKW   1
JSTACK:  .BLKW   1
 .=.BLK.+200
LOOP:
 .=.BLK.+362
 JSR     PC,PAT013
 NOP

 .=.BLK.

 .PSECT  $$PAT
PAT013:
 MOV     I,R0
 ADD     R0,R0
 SUB     ISTACK,R0
 CMP     JSTACK,R0
 BLE     10$             ; This instruction was BGE 10$
 MOV     $QSTAK-2(R5),$QSTAK(R5)
 RTS     PC
10$:
 TST     (SP)+           ;CLEAN UP STACK
 MOV     $QSTAK+198.(R5),$QSTAK+200.(R5)
 MOV     I,R0
 DEC     R0
 MOV     R0,$QSTAK+198.(R5)
 ADD      2,R0
 MOV     R0,$QSTAK(R5)
 JMP     LOOP

 .END



 .TITLE  LBNORD          ; FOR V4.0 BRU
;
; MODIFICATION:
;
;        01.3 -- FIX SORT LOGIC ERROR IN LBNORD (21 MAY
1982)
;                THIS IS PART OF THE REAL FIX FOR
5.1.17.9
;                C. T. MICKELSON, GOODYEAR AEROSPACE
```

```
CORP.
 ;                 AKRON, OHIO 44315 (216) 796 - 2388
 ;
 ;                 ALSO, REMOVE "FIX" OF 5.1.17.9
 ;
 ;         01.4 -- FINISH CORRECTION TO QUICKSORT BY
STACKING
 ;                 LARGER PARTITION OF LIST RATHER THAN
ALWAYS
 ;                 THE LEFT PARTITION THIS ELIMINATES
 ;                 POSSIBILITY OF A STACK OVERFLOW ERROR
BUT MAY
 ;               MAKE PAST BRU TAPES INCOMPATIBLE.
 ;                 C. T. MICKELSON, GOODYEAR AEROSPACE
CORP.
 ;                 AKRON, OHIO 44315 (216) 796 - 2388
 ;
  .PSECT
  .BLK.=.

  .IDENT  /01.3/
  .=.BLK.+56
  MOV      20.,R2          ;Remove "FIX" of 5.1.17.9

  .=.BLK.+472
  DEC      $QSTAK+200.(R5)

  .IDENT  /01.4/
  .=.BLK.+12
 I:        .BLKW   1
 J:        .BLKW   1
 ISTACK:   .BLKW   1
 JSTACK:   .BLKW   1
  .=.BLK.+234
 LOOP:
  .=.BLK.+416
  JSR      PC,PAT013
  NOP

  .=.BLK.

  .PSECT   $$PAT
 PAT013:
  MOV      I,R0
  ADD      R0,R0
  SUB      ISTACK,R0
  CMP      JSTACK,R0
  BLE      10$             ; This instruction was BGE 10$
  MOV      $QSTAK-2(R5),$QSTAK(R5)
  RTS      PC
 10$:
  TST      (SP)+           ;CLEAN UP STACK
```

```
MOV      $QSTAK+198.(R5),$QSTAK+200.(R5)
MOV      I,R0
DEC      R0
MOV      R0,$QSTAK+198.(R5)
ADD       2,R0
MOV      R0,$QSTAK(R5)
JMP      LOOP

.END
```

I would now like to turn to a discussion of the compatibility problem that was introduced into BRU when SPR Article 5.1.17.9 was published. The following observations can be made about the operation of LBNORD as distributed by DEC:

1. Given an "unlimited length" stack, the algorithm as distributed will always sort a list of retrieval pointers. Proper performance of the algorithm is assured regardless of which partition (right or left, larger or smaller) is pushed onto the stack, so long as the stack does not overflow! This is true due to the fact that during each iteration of the algorithm, after a pivot element has been properly positioned, all elements to its left are less than, and all elements to its right are greater than the pivot. Once the pivot is properly located, the stack is used simply to remember the upper and lower boundaries of one list partition while the other list partition is completed. Stacking the "wrong" partition (read left side only in DEC's distributed version) has no effect on which partitions are generated. Stacking the "wrong" partition only effects the order in which partitions of the list are generated. Therefore a true sort is maintained. The only reason to stack the larger partition is to guarantee that the deepest stack penetration is limited to log base 2 of the list length (or 10(10) for BRU).

2. The effect of the "typographical" error in the original version of LBNORD on the ordering of data on tape is a little more difficult to understand. This error was originally discussed in the July Multi-Tasker. Briefly summarized there is a flaw in the distributed version of LBNORD when modifying the partition stack when the pivot index reaches the right end of a partition. The algorithm intends that the value I-1, designating the upper bound of

the left list partition, be saved on the stack. The implementation saves the value of I, and decrements a location in memory 50 words away from the location in which I was stored! This location is in the upper half of the stack of right end indices for saved list partitions. The failure to properly decrement the value of I simply includes a previously positioned pivot element in the list partition placed on the stack. However, it is known that all elements to the left of this point are already less than the included pivot element. Therefore, including the pivot element in the list partition will not change the final sorted order of the list. The more serious flaw is decrementing the wrong word in the stack. If the stack never reachs 50 levels deep, no active element in the right end index stack will be changed, so the list will still be sorted properly. Fortunately, for a sufficiently "out-of-order" list (see discussion below) the frequency of occurrence of such a deep stack is rare.

3. The distributed version of LBNORD makes a pre-sort pass over the list of retrieval pointers to determine if the pointers are sufficiently "out-of-order" to warrant sorting. The initial "out-of-order" count used to make this sufficiency decision was 20(10). The recommendation of SPR 5.1.17.9 is to raise this count to 40(10) or higher if stack overflows are still encountered, (i.e. the stack is not "unlimited" enough for the particular list of retrieval pointers encountered by BRU). The effect of this change is to place more sets of saved data, (those that satisfy 20 <= "out-of-order" count <= 40) on tape based upon unsorted retrieval pointers. If read back with a version of BRU using a different "out-of-order" count, proper restoration of data cannot be guaranteed. This is because a different "out-of-order" count version of BRU will sort retrieval pointers for some sets of data that were not sorted when written to tape.

4. 4. The patches provided in this article will not change the order of data written on tape. The patches simply prevent stack overflow by limiting stack growth to the theoretical maximum for quick-sort.

There is unfortunately one additional complication in this analysis. This is due to the method used in DEC's version of LBNORD to perform exchanges of retrieval pointers within the sorted list. Each retrieval pointer in the list is comprised of five words. When an exchange of pointers is to be performed, one pointer (5 words) is stored on top of the stack while the second pointer is moved into the space occupied by the first. Finally, the saved pointer is moved to the space occupied by the second, completing the exchange.

This use of the stack is perfectly acceptable for a BOUNDED stack with at least five additional words for the temporary data, but due to the organization of the stack in memory, and the nature of DEC's implementation of quick-sort, the stack depth can reach or exceed 96(10). When this occurs, moving a retrieval pointer onto the stack will destroy up to five words of the second half of the stack. However, the distributed version of LBNORD does not detect stack overflow until the stack depth exceeds 100(10). Therefore it is possible to corrupt the stack without overflowing it! This occurrence should be just about as rare as the stack overflow in sort error treated in SPR Article 5.1.17.9. In fact, if this type of corruption does occur, it is likely that BRU will odd address trap or violate memory protection and crash. This was probably the mechanism that caused BRU to crash last May while saving our disc, and led to my discovering the typographical error documented and patched in the July Multi-Tasker.

In summary, a version of BRU with a given "out-of-order" count, and with the patches provided here applied should be able to restore most any BRU tape successfully written with a version of BRU having the same "out-of-order" count. This, coupled with the fact that it is now impossible to overflow the stack, can solve the incompatibility problems of BRU tapes (between V3.2 and V4.0) by using a universal "out-of-order" count. The value 20(10) is suggested, as this is probably the most widely used due to its initial distribution.

EDITORS NOTE

After having received Carl's article, and just as I was about to send this issue to the DECUS office for printing I received the following from Carl.

Enclosed is a copy of DEC's response to my last BRU LBNORD SPR. It effectively confirms my solution to the sort module problems in BRU that have existed for so long.

The DECUS Atlanta patches referred to in this DEC response were discussed in my Sept/Oct 82 Multi-Tasker article. Regardin the availability of future corrections for BRU, I have been told by the BRU maintainer that V3.2 BRU will no longer receive published changes. Thus we can expect that Software Dispatch Articles 5.1.17.2 (Feb 80; BRUBITMAP), 5.1.17.16 (Apr, 81; SCNDIR), 5.1.17.18 (Jul 81; BEGIN and BRUALLOC), and 5.1.17.21 (Dec 82; 11 assorted cumulative patches) represent all of the known and to be published patches for V3.2 BRU. These patches, together with those published for LBNORD, represent a version of V3.2 BRU that should run very reliably, at least regarding retrieval pointer sorting.

I would like to discuss some of the problems I encountered in applying the BACKLB patch published in 5.1.17.21, Dec 1982. The baseline *.PAT files were originally acquired from the RSX-11M Autopatch E distribution. Using these patch files as a starting point, the new changes were added to BACKLIB.PAT from pages 1-6 of the subject article. The following difficulties were encountered:

1. On page 5, at line 260$:, the macro call RESRG should read JSR R5,RESRG.

2. On page 3, after MOV $LAB1+4,-2(R2), the line .=.BLK. is missing from the Autopatch E baseline file. The checksum /CS:125544 on page 51 of the patch article is correct for the changed patch file, without this .=BLK. line. However, without this line, the patches to the blank PSECT made for IDENT 1.07 on pages 4 and 5 are put into the module at the wrong place, because, without the line, the assembler's base for this PSECT has been lost. Adding the missing line .=.BLK. after the MOV $LAB+4,-2(R2) instruction, places the patchescorrectly, but changes the checksum for the patch file to /CS:125764. One wonders whether this particular published patch was ever applied directly to an object module, or whether the patch file was developed from source changes only.

3. Finally, the line .BLK.=. after MSGFLG=.BLK.+1576, on page 5 of the patch, should read .=.BLK. to preserve the base address for the blank PSECT for subsequent patches. This does not change the checksum; it remains /cs:125764.

The checksums for the remaining modules and their patches are correct as published in the Dec 82 article.

SPR NUMBER 11-46577A

PROBLEM STATEMENT:

The user submitted his solution for the BRU overflow problem in the module LBNORD.

RESPONSE:

Thank you for the time you spent helping to solve this BRU problem. We basically agree with your last article in the Multitasker, except on one point. BRU version 4.0 does not produce incompatible tapes because we have a special flag word at the beginning of the data area on the tape. This word is equal to 1 if retrieval pointers for the data were sorted, and 0 if the retrieval pointers were not sorted. This means that during the restore operations, we do not depend on the sort parameter and cna always determine when we have to sort the retrieval pointers. The problem appears only when we are restoring BRU Version 3.2 tapes. In this case, we definitely need to use the same sort parameter during the backup and restore operations.

The articles in the AUGUST 1980 Software Dispatch, and one that we distributed at Decus, represent a possible solution to the reported problem. Of course, it is not the best possible solution, but it always works. This is the main reason we still recommend it to our customers as a temporary workaround until we publish a new correction which will eliminate this problem.

RSX-11M to RSX-11M-Plus Migration

Allen A. Watson

Bergen Evening Record

150 River Street
Hackensack, NJ 07602

1.0 REASONS FOR GOING TO M-PLUS

1.1 Shadow Disk

Our main reason for migrating from RSX-11M to RSX-11M-PLUS was the added capability of what is called "shadow disk". We have been, and still are, developing several large applications that will be handling volatile and sensative data: an advertising accounts receivable package, a display advertising layout package, and a newspaper circulation package. All of this data is changing rapidly from day to day. Some of it, such as the ad layout data, is critical to the daily production of the newspaper.

In the newspaper computing world we have a saying: "THERE IS NO TOMORROW." In most computing applications in the event of disaster it is possible, although not desirable, to "do it tomorrow". Getting out payroll a day late may be expensive, it may cause alot of discontent, but in the worst case it can be done. Producing Tuesday's newspaper on Wednesday, however, simply cannot be done. For a newspaper, there is no "tomorrow".

For applications involved in the production of the paper, therefore, data backup is imperative, and recovery in case of disaster, such as a head crash, must be nearly instantaneous. Typically, in newspaper applications, five minutes is considered a maximum recovery time. It may be nice in movies about newspapers to have someone screaming "Stop the press!", but in real life stopping the press is about the last thing you ever want to do. To stop the press because the computer has lost data necessary to the production of the next pages would cost so much that you could buy a complete secondary computer with the money. In fact, almost every paper I am familiar with has totally redundant computer hardware for just that reason.

What we needed was dual recording of critical data on two identical disk packs , so that if a head crash occurred, all that would be necessary to recover would be to boot up from the second disk. That is what "sha-

dow recording" is. A complete, up-to-the-second, mirror image copy of your disk is maintained on another disk mounted on another drive.

We briefly considered a transaction log type of system, where any updates to files would be logged onto a disk or a mag tape. This required a lot of support software: each application would have to handle its transaction logging, programs would have to be written to recover from transaction logs and periodically purge them, and so on. Any new application would have to include support of transaction logging, making all the applications larger. We rejected this approach in favor of M-PLUS and shadow disk because shadow disk is transparent to the application and requires no additional code.

A nice side benefit of shadow disk from the viewpoint of operations is that it eliminates the need to copy the disk you are shadowing; you have a constant, on-line copy.

In our brief experience with shadow disk, we have found that it adds little overhead to the system, especially when you funnel the shadow disk through a separate controller using a unique ACP.

Remember, however, that if you have four disks you want to shadow, you will need eight disk drives! If the ability to recover from a head crash in five minutes instead of an hour, or much more (you know what it would take at your site) is worth the price of another drive, you should consider using M-PLUS shadow disk.

At THE RECORD we have experimented with combining the Virtual Disk package from the RSX SIG tapes with shadow disk. This allows us to designate selected sets of files that have been assigned to virtual disks, which are located on several disk drives, for shadowing on a comparable set of virtual disks that have been located on a single "shadow" drive. We have limited the files to be shadowed to the critical few, instead of duplicating entire disk packs, and have been able to reduce the number of drives needed to shadow those files. I will be discussing this aspect of shadow disk in a separate session Thursday afternoon at 4:30.

1.2 Overlapped seeks for disk

M-Plus supports overlapped seeks for disk. This feature allows multiple disk units attached to a single controller to perform seeks (head movement) simultaneously, although only one data transfer can occur at any one time. Most advanced disk controllers support this feature, as ours do. Since we anticipated an operations environment where up to eight drives might be accessed through a single controller, it seemed to us that overlapped seeks would be a considerable help to disk throughput.

In our current operations we rarely have more than two drives operating through a single controller, and in most cases we are operating on just one drive, so we have no concrete data yet on savings through overlapped seeks.

1.3 Supervisor mode libraries

Available only on 11/70's and 11/44's under M-PLUS, supervisor mode libraries are resident libraries that double a user task's virtual address space by mapping the instruction space of the processor's supervisor mode.

That was a near quote from the Executive Reference Manual. For those of you who haven't yet learned to speak DEC, let me give a rough translation. A task or program under RSX-11M is only allowed to have 32K (roughly 32,000) locations of memory to run in because the hardware can't count any higher than that. Supervisor mode is like a second counter, allowing another 32K locations to be used. For a little memory overhead in your program and some run time overhead when your program has to switch modes to get at the stuff in the supervisor mode library, you can double your program size.

That was important to us because we are converting a number of programs from an IBM 370 with virtual memory to run on our 11/70. Most of them were just too big to run without overlaying and breaking them up into subtasks, with the attendant overhead in increased execution time. Supervisor mode libraries give us the ability to build bigger tasks.

In addition, many of the DEC utilities can be built

using a supervisor mode library for File Control Services (FCS). That makes those commonly used programs smaller, and for the larger ones that require a lot of overlaying under RSX-11M, allows DEC to reduce the overlaying, thus reducing the number of overlay calls from disk. In sum, the utilities are smaller and run faster. The option of building FCSFSL utilities is offered during SYSGEN and I advise you to take it.

1.4 Secondary pool

Under RSX-11M we had frequent system crashes when we ran out of pool. For you new users, "pool" is a space in the executive used by the system as a work area to contain data structures such as system lists, control blocks, and I/O packets. Every file that is open has a file control block in pool; each installed task has a task control block; active tasks have task headers in pool; each terminal has a user control block, and so on. There's a lot of stuff in pool and only a limited space in the Exec. When it fills up, the system crashes.

How many of you RSX users have experienced that problem?

Before M-PLUS we tried lots of tricks to get more pool or to keep from crashing. We installed the Pool Monitor Task from the SIG tapes; we put in patches from Jim Downward at KMS Fusion that allowed us to run with fewer tasks installed; we put in a patch to the terminal database to reduce the number of SCB's for terminals. Each thing helped a little, then we'd run out of pool again as the load increased.

Under M-PLUS we have yet to come close to running out of pool space. One of the main reasons is secondary pool. Secondary pool is a memory partition that is outside of the Executive, and it can be as large as you want. M-PLUS uses it for more permanent or less frequently used data structures, and thus frees up that space in primary pool. Task headers, for example, go there, so under M-PLUS there is almost no limit to the number of tasks you can install.

We had a pool problem under M; no longer. We come up with over 11,000 words of primary pool. We'll run out of memory long before we run out of pool. PMT seems almost superfluous under MPLUS. But we can (and will) buy more memory! and you could never buy more pool.

1.5 Directive common

Directive common is another way you get more pool space; some of the executive directives are moved into a common partition thus freeing up more space in the exec.

1.6 Multi-user tasks

Our system has lots of users. We anticipate having up to 52 terminals on a single 11/70. That many users can fill up memory with tasks awfully fast. M-PLUS allows the building of multi-user tasks, in which a single copy of read-only portions of a task is shared by many users. Even on our development system, the multi-user versions of EDT and PIP have helped reduce checkpointing in the system. We are making some of our application tasks multi-user also.

1.7 Multistream batch processing

We wanted batch processing so that users could schedule long, time-consuming tasks to be done in off hours. Batch allows you to do this. In effect you submit a command file to the queue manager just like a print job: SUBMIT MYJOB/AFTER:17:00. Also, when development gets really heavy we can ask users to submit jobs to batch streams rather than running them directly, thus limiting the number of simultaneous compiles and task builds.

1.8 Task and user accounting capability

We expect our system to overload before long even under M-PLUS. When that happens we will be able to use the resource accounting facility to determine which tasks are overloading the system, and with what kind of activity: CPU, disk, QIO's, etc. It is possible we may develop a charge-back system to our user departments.

2.0 TRANSITION PROCESS FROM M TO M+

I would say we had a rough transition, but largely because of two factors: first, we elected to migrate at exactly the time DEC was discontinuing Version 1.0 and starting on Version 2.0. For several months we couldn't get either version from them. Finally they delivered Version 1.0; about six weeks later, we got Version 2.0. We had barely adjusted to Version 1.0!

Second, we had all non-DEC disks and disk controllers. It's very hard to SYSGEN a system when you can't run any of your disks on it. We ran our Version 1.0 GEN from our RSX-11M 3.2 system, and then GEN'ed our Version 2.0 using the Version 1.0 system. Before starting the GEN, we had to modify SAVE, the DB driver, other utilities and the SYSGEN command files to compensate for our foreign controllers. It was frustrating because we spent weeks debugging code for the disk sub-system without being able to see the new system at all. We read the manuals and dreamed great dreams; meanwhile we couldn't even boot the thing.

If I were back buying our system in the first place I would include at least one standard DEC disk system for doing SYSGEN's if for no other reason.

One nice thing is that DEC had included very clear instructions on how to GEN a Version 2.0 system from a Version 1.0 base -- just what we needed. They even have a special command file for doing it, letting you build the Version 2.0 versions of MAC, TKB and IND that you need for the GEN on your Version 1.0 system. We did it and it works. In general the Version 2.0 SYSGEN manual is much clearer and the SYSGEN procedure is much simpler than any I have done before.

Autoconfigure is a new thing DEC has added that eliminates the messiest part of a SYSGEN: specifying CSR's and vectors for all your hardware. Autoconfigure just goes out there, probes around the system to figure out what hardware is attached, and generates all that stuff for you. It then allows you to examine and edit the results in case it goofed or left out something it couldn't recognize. If you have a system put together by DEC with mostly DEC hardware, use autoconfigure; or if you know your CSR's and vectors meet their standards, use it. Our CSR's and vectors are definitely non-standard even for the DEC gear (the system was put together by an OEM) and we found it more convenient not to use autoconfigure, so I can't say how it works.

3.0 PROBLEMS ENCOUNTERED IN SYSGEN

3.1 Bug in Building the Executive

The command file SGNBE.CMD has a bug in it that DEC knows about. When the exec finishes task building the command file attempts to wait for the cross-reference to complete with the statement:

```
  .IFINS CRF... .WAIT CRF...
```

For some reason this causes an error message saying "SPAWN FAILURE" and the SYSGEN command file aborts.

The only thing that fails is the indirect command file. The task build of RSX11M has worked fine. All you really need to do is make sure CRF has finished and continue with the GEN. I checked SGNBE.CMD and the only remaining executable line was "TKB @DRIVERS". So I typed that in directly, waiting for it to complete, and then re-started the SYSGEN at the next phase after "Build the Executive and Drivers". Worked fine.

You could modify SGNBE.CMD (see note on SYSGEN command files below under "Hints from our Experience") by commenting out the line at fault and replacing it with a .PAUSE. When it pauses, just check active tasks. If CRF... is not active, then resume. DEC software support suggested removing CRF... before starting the build of the exec, but that seemed counter-productive to me.

3.2 Getting correct versions of N.P. tasks

Pay close attention to the NOTE on page 3-63 of the SYSGEN manual. If you make any changes to the task build files to select options on some of the non-priveleged tasks, and you fail to move them from [1,54] to [3,54] after they are built, you won't get your tailored versions of those non-priveleged tasks when you bring up the system. The versions of those tasks from the distribution tape are in [3,54]. The versions you build, for some reason, go into [1,54]. When SYSGEN creates SYSVMR.CMD file it specifically installs non-priveleged tasks from [3,54] so you get the DEC originals, not the ones with the options you so carefully chose. DEC should build non-priveleged tasks

into [3,54] if that's where they expect them to be, and I have complained to that effect in an SPR.

4.0 HELPING THE USERS ADJUST

Some of these things probably apply equally well to new users of RSX-11M Version 4.0. In general the transition for the users of our system was painless, but not without some effort ahead of time and behind the scenes.

4.1 Getting used to DCL

For someone coming to M-PLUS from M Version 3.2, DCL is something new. We elected to put in DCL as the primary Command Line Interpreter (CLI) and to modify the task build file (see below) to select the option to allow any unrecognized commands to fall through to MCR. If you don't do that, any user of DCL is going to be extremely annoyed when typing PIP fileA=fileB/RE elicits an "UNRECOGNIZED COMMAND" message. You can't get directly at anything through DCL, not even PIP, unless you enable the fall-through-to-MCR option in the task build.

Overall falling through to MCR works fine; people used to MCR just go on using it as they always did and never know DCL is there until they get around to reading the new manuals. Almost.

There are a few problem spots where MCR and DCL use the same command, as in SET, MOU, MAC, FOR, and INS. A user enters a familiar command like "MAC FILE=FILE" and is given the error message:

```
  MACRO -- Extraneous input
  MAC FILE=FILE
```

That's DCL complaining, because its syntax is different. Simply alert your users that if familiar commands don't work, they should try them with a dot in front of them. For example ".MAC FILE=FILE" will work just fine. The dot tells DCL to schlep the command off to MCR without even looking at it.

If they're going to do a lot of that sort of thing, they can set their terminal to MCR as primary CLI with SET TERMINAL MCR. In general, SET TERMINAL MCR and SET

/DCL=TI: should be the first two new commands you teach your users if DCL is in your system.

4.2 Command file execution

If a terminal is running under DCL many old command files won't work if they use MCR commands like "MAC FILE,FILE/-SP=FILE" because "MACRO" is a DCL command expecting DCL syntax. I happen to like DCL, so what I do is put this into my command files:

```
  .setf DCL
  .if <CLI> = "DCL" .sett DCL
  .ift DCL SET TERM MCR
          :
          :
    contents of old command file
          :
          :
  .ift DCL SET /DCL=TI:
```

That could be more generalized if you plan on multiple CLI's:

```
  .sett MCR
  .enable substitution
  .sets CLI <CLI> ! Save starting CLI !
  .if CLI NE "MCR" .setf MCR
  .iff MCR MCR SET /MCR=TI:  !Assumes MCR command in all
CLIs!
          :
          :
  .iff MCR SET /'CLI'=TI: ! Restore starting CLI !
```

4.3 Hints to KMS CCL users

We still use CCL. It works fine. We have three commandline interpreters: DCL, MCR, and CCL. DCL comes first; any unrecognized commands fall through to MCR; finally, CCL is installed as CA. (catch-all) to handle anything MCR does not know. All I did was to rebuild CCL.TSK on the new system, and since CCL is not priveleged even that was probably unnecessary. None of the Jim Downward patches to system routines were made; most of them exist in MPLUS as distributed. One Downward patch to MCR we miss was the one that forced it to pass everything to CCL (instead of kicking out things like

"?" and "LI"). When we want to add a new command it is still ever so much easier to edit two or three lines into SYSCCL.CCL than to master the complex syntax required to build a DCL command table entry, edit the file, assemble it, and double task build DCL!

The /CMD parameter to RUN is not exactly like the old KMS /PRM parameter: it expects the task name in the first four characters. In other words, it clobbers the first four characters you pass. You must say:

```
          RUN $MAC/CMD="MAC FILE=FILE"    instead of
          RUN $MAC/PRM="FILE=FILE"
```

The CMD parameter does not exist for INSTALL, as PRM did in the KMS mods. One strange annoyance is that you cannot pass parameters to an installed task from the RUN command <u>using CMD</u>; to use RUN/CMD the task must be <u>non</u>-installed.

4.4 New Introduction manual

The Introduction To RSX-11M-PLUS book with accompanying files in [200,1] are a great training aid for new users. I did find some of the examples in the book did not work; some files used in the text were missing, but were easy to provide. Mostly they were one-line text files used to illustrate several commands like TYPE and PRINT. Nothing major was missing. I suggest you run thru the book yourself before handing it to a new user, and fix the things that don't work. Even your old hands should work thru this because they will learn DCL and other stuff they probably never knew or have forgotten (e.g. EDT line edit mode for people who always use keypad).

One outright error: the manual says the EDT command "T LAST" types the last line of a file; in fact it types the last line referenced. The command should be "T END-1".

4.5 Queue Manager and Error Logger

The Queue Manager and Error Logger commands have changed entirely from anything you ever knew. God knows they needed it, but be warned and spend some time looking through the documentation before turning the system loose on your users. You may want to write some memos to frequent users and computer operations personnel, or hold some re-training classes. At least alert them to the excellent HELP files and how to find them.

If you have any command files that used to make error report generation easier , throw them out. They'll be useless. Be prepared to rewrite, and to find that it is easier this time. I wrote a file called ERRORS.CMD for our operators to use (it will be in [333,100] on the SIG tape) that walks them through most of the options.

The canned report option for the error log REPORT GENERATOR did not work. You are supposed to be able to enter:

```
  RPT /RE:DAY
```

to get a full summary report on all today's errors, for example. It bombs. My ERRORS.CMD file could not make use of this switch, unfortunately. I have SPR'ed this.

5.0 HINTS FROM OUR EXPERIENCE

5.1 Multi-path access to disks

Setting up procedures for proper handling of dual-ported disks, disks that in addition are accessible from two or three CPU's simultaneously, was very complex. We have three CPU's, three controllers, and six drives. The disks are each connected to two of the three controllers, and all three controllers are connected to all three CPU's. There are two paths to each disk from each CPU.

M-PLUS allows you to specify disk drives as dual-ported. Since that is what we have physically it seemed to make sense to tell the software about it. It took us weeks to find out all the things that could go wrong. We found no actual bugs in the software, just mass confusion for us as users.

One rule of thumb: DON'T LIE TO THE SYSTEM. If you SAY there are two paths to a disk there damn well better BE two paths. You see, we have this nice switch panel that can put individual ports on a controller off-line... Well, if the second path for DB2: is switched offline, and you try to MOUNT DB2:, mount will time out. If you're booting from DB2:, tasks will start getting

load failures in STARTUP right after CON ONLINE ALL is executed (more on the CON tasks below).

We also found (with CDC drives and SI controllers) that dismounting a pack on one CPU could knock it off-line for another CPU as well. We had one CPU booted from DB5:. A user on a second CPU mounted DB5:/NOWRITE (that's a new switch on MOUNT that is very useful), and when he was done, he dismounted DB5: like a good user should. Bang! down goes the first CPU. For some reason, even though our drives look like RP04's to RSX, the dismount command is trying to unload the disk! And for some reason the disk seems to recognize it even though it can't be unloaded from software. (Could be a hardware bug.)

We finally discovered that, to be safe, we should mount disks with a new switch, /LOCK=N, which sets the default for dismount to "no unload". That way users don't have to remember to use DMO DBn:/LOCK=N. We set up a MOUNT.CMD command file and have everyone use it to mount a disk so the right switches are always used.

Another switch useful in multi-path situations is /LRU=0. Whenever we mount a disk /NOWRITE we also add /LRU=0, which instructs RSX not to cache directories in memory but to always read from the disk. That may sound inefficient, but when we mount a disk /NOWRITE it is usually because it is mounted for writing from another system. Directories can get strangely out of sync when one system is writing to a directory and another thinks it has the directory cached in memory!

What we really need is a mechanism that completely prevents a disk being mounted for writing from two systems at once. We have done that, of course, and the result is hundreds of multiply allocated blocks as each system blithly writes using its own copy of the bitmap of free blocks. One useless disk and lots of grouchy users. We are currently working on some mods to MOU and SAV to accomplish this.

5.2 CON, the System Reconfiguration task

The System Reconfiguration task is both an enormous boon and a colossal pain. This task enables you to place devices in your configuration either on or offline by software command: CONFIGURE OFFLINE DB0:, for example. You can also display and even change CSR's and vectors! This affords great flexibility. But there is a whole new command set to learn.

The pain comes from what I said above: DON'T LIE TO THE SYSTEM. RSX-11M users are used to being able to do peculiar things like spinning down one disk and spinning up another in its place without dismounting it from the system. M-PLUS won't let you do that; the minute you touch the button it dismounts the drive automatically. This particular protection is, I suppose, good. But sometimes I want to lie to the system! Like when I blow a home block on an otherwise good disk and want to recover its files. I've kept a Version 3.2 pack around just so I can play that game.

A related difference is the fact that under M-PLUS any access to mag tape requires that the tape be mounted. To initialize a tape you can't just allocate yourself the drive and initialize; you allocate it, MOUNT it /FOREIGN , and then initialize. Same with BRU and other utilities that used to access unmounted tapes and disks under RSX-11M; no more, you have to MOUNT /FOREIGN.

In general M-PLUS forces you to be more careful about what you do with the hardware, and to tell the system using CON before you do it. If we are going to switch an access path to disk offline, we first must CON it OFFLINE. Makes sense, but at first it creates strange situations for someone used to M.

5.3 Task Build options

Before building non-privileged tasks, I recommend you search thru all xxxBLD.BLD files in [1,20] for all the GBLPAT and GBLDEF lines to see for yourself what options you have in task building the various utilities. These are command files used by SYSGEN to create xxxBLD.CMD files in [1,24] for the related task builds. For example, PIPBLD.BLD contains switches to select options for PIP.

There is a point where SYSGEN pauses after creating the command files in [1,24] and asks if you want to edit any of them, and you could wait until then and then do your editing. However, if you re-run SYSGEN for any reason you'll have to repeat the editing. In addition, you can, if you wish, build three versions of many utilities, one regular (overlaid), one using FCSRES, and one using FCSFSL. Then you have to edit three command files in [1,24]. If you edit the xxxBLD.BLD files in [1,20]

before starting the non-privileged task part of SYSGEN, then your options will automatically be included in all three versions every time you do a GEN.

5.3.1 PIP

For example, PIP is advertised as having the option to preserve creation date on copies, but nowhere does it tell you that you must edit a global in the PIPBLD.BLD to obtain this option.

5.3.2 INDIRECT

The Indirect Command processor also has nifty options. For example, you can have it default to the system UIC (or another you designate) if the command file is not found in the user's UIC; once again, however, you must edit the Build file to get this. IND is now called ICM just to confuse you.

5.3.3 DCL

DCL has a couple of options, most useful of which is allowing unrecognized commands to fall through to MCR. Edit [1,20]DCLBLD.BLD to get it.

5.3.4 Other tasks with options

Tasks I found having build file options:

DMP	CMP	AT.	LPP	MAC	PIP	PRT	QCL
QMG	RMD	SHA	1KB	ACN	BYE	HEL	LBR
RPT	SAV	DCL					

6.0 MISCELLANEOUS COMMENTS

The ability to broadcast to users by name is nifty: finds them even on multiple terminals.

The sample DTR reports for Resource Accounting are excellent tools for producing your own customized reports. All of the accounting records are now accesible through Datatrieve.

In general, Mplus is easier to manage for an unsophisticated user because it translates more of what you need to know into human-readable form.

Be aware that under M-PLUS a task named "...XXX" NEVER RUNS; it is a prototype task only. When TTO runs it by saying "XXX", it gets a task called "XXXTO". If you have any command files that try to do ".WAIT ...XXX" they will no longer work. Such tasks are not intended for multi-terminal use and should be installed with task names not in the form "...XXX".

SIG programs we built under MPLUS (versions on SIG tape): SRD (multiuser, non-overlaid), TECO, UIC, LIST, GREP, COOKIE, RNO, DOC, DUNGEON, C(XCC), PREDAY, TYPE (renamed TIPE), TREK, TCF, BRUDIR, SRDCMD (CMD), ADVENT, RMC, LUT, USERMN, TRUNC, RATFOR, PACMAN. Most required no modification, only a few required other than minor mods to the build files.

The HELP files on Version 2.0 are fantastic, in some cases more accurate than the manuals. I wrote three TECO macros, HDX, HFL, EDH, and TEH (all with both TES and TEC extensions), to help me step through all of them and index what is there. See [333,100] on the current RSX tape. (The text of this handout -- possibly with updates after 15-OCT-82 -- will be there, too as M2MPTALK.DOC.)

The Night of the SIG Tape

Roger Jenkins

Wycliffe Bible Translators
Huntington Beach, CA 92647

At the Fall Anaheim Symposium, several of us were standing around waiting for others to show up before we left to create the SIG tape. Ralph Stamerjohn walked by and made the suggestion that we should place an article in the MULTITASKER describing the work that goes into

----

the SIG tape collection. This article is a result of that suggestion.

At the Spring 1982 Atlanta DECUS Symosium, I had approached Jim Neeland, RSX SIG Tape Coordinator, and had offered our site as a collection point for the Fall Anaheim Symposium. Our site is about 12 miles from the Disneyland Hotel, so it seemed like it would be a convenient point from which to create the SIG tape. Jim accepted my offer, so Tuesday night around 10:00 PM we loaded seven people and a box of 30 tapes into two small cars and drove to our site for a marathon tape copy session.

The first step (after the usual site inspection every programmer performs when he walks into a new machine room) was to prepare a scratch disk onto which the contents of all of the tapes would be placed. We have three RP06 equivalent drives and two tape drives. I had requested exclusive use of the machine, so we had plenty of resources.

Next we had to load several utility programs: TECO (mine was not up to date), version 3.2 BRU, TPC, FFL and perhaps there were others that were necessary to create the SIG tape.

Before long there was quite a flurry of activity. Charles Goodpasture and Vince Perriello were carefully checking tapes and release forms to make sure that each tape had a release form and that it was signed. Jim Neeland and Glenn Everhart were copying the software from tape to disk, John Osudar was mounting tapes left and right and Tony Scandora was the "scribe" recording the directories into which each tape was placed.

I asked Jim, who was also coordinating the overall activity, how he determined which UIC the contents of a tape went into. Basiclly there is a set of 4 rules:

1. If the person submitted a tape last year, his data went into the same UIC as last year.

2. Each LUG is assigned a group number, and new submissions are placed into subsequently numbered member UICs within the LUG's group.

3. If a person is not part of a LUG but is close enough geographically to be a part of one, his submission may be included as part of the LUG group.

4. For every one else, they are placed into a common group with each submission getting its own member number.

If the tape UIC was wrong, then it was copied into the right UIC when possible. At least one tape was in BRU format and we wanted to place the files in a different UIC than they were in on the tape. This involved copying them to the same UIC as on the tape, but that UIC on the disk already had files in it. So first the files on the disk were renamed to a different UIC then the tape was copied and those files renamed to the right UIC then the original files were renamed back in to the correct UIC. Now you know why we were here until 6 AM.

The copy process itself was no simple task. Even if the submission form told the tape format and was correct, then several tries might be necessary. For example, if the submitter failed to specify whether 3.2 BRU or 4.0 BRU was used, then one BRU would have to be tried and if it didn't work the other would be tried. A lot of intuition and guess work was required if the format was not specified. We would try FLX, ANSII PIP, 3.2 BRU, 4.0 BRU and even DMP! Jim tells of one time when a format even turned out to be RMSRST. ("It mounted like an ANSII tape but PIP wouldn't read it!")

After several of the tapes had been loaded to disk and all of the submission forms had been checked, Charles and Vince switched to mounting the tapes (they did the "reel" work). Then John began the laborious task of reading all of the README files and summarizing them. For each UIC, John created a file from README info containing a one line description of each program. Then these files were sorted in UIC order and merged to form the final SIG tape contents summary. This data was then edited further and placed in [300,1]RSXF82TPE.DOC.

When the disk was finally ready, we made copies of the preliminary tape for each of us. Jim then took that copy back to his site to prepare the final tape. He will notify the members of the distribution tree by mail when the tape is ready. It is then the responsibility of each LUG librarian on the tree to contact his parent node to get a copy of the tape. If for some reason his parent node can't him a copy of the tape, he should contact his grandparent node.

The tape copy team had several suggestions for submitters that would make the tape copy job easier:

1. Submitters from LUGs should get a UIC number from the LUG librarian.

2. If you have to use a specific UIC, use one below [300,1] to avoid colisions with other submissions that may already be on the disk when your tape is copied to disk. Label the tape and submission form with the UIC you are using.

3. FLX is the easiest format to work with for small submissions and especially when you don't know a UIC number. If you use BRU, specify the backup set name(s) and the UIC(s) used.

4. Always include a README.1ST (not README.DOC, README1ST.DOC, etc.) The README.1ST should be a summary of the programs in the UIC (one or two lines per program), not the main documentation.

5. Be sure to fill out the submission form correctly. Tell the truth about the tape format and density.

I think that the SIG tape is one of the most sucessful "products" of the RSX SIG. I know that my site has benifitted greatly from it. We all owe our appreciation not only to those who have contributed to it but also to Jim Neeland and the others who have worked long, hard and late hours to put the tape together over the years.

Unlimited Keywords in TPARS

Ken Cross

Perceptics Corporation
221B Clark Street
Knoxville, TN 37921

TPARS is one of the neatest packages that DEC has made available to users. Those unfamiliar with it should check it out in the I/O Operations Manual. It is a little strange at first, but once you get the hang of it, you find many useful applications.

We have used TPARS to develop the operator interface for a Falcon that controls the image processing system we have developed. It provides very powerful command syntax structures, but was restricted to 64 keywords. I had run into this limitation on other projects and got around it in very complicated ways in my application program. This time I decided to fix it right.

The fix involves moving the address of the keyword strings into the "extension" word of the transition tables. It doesn't take any more memory than before and simplifies the processing. All keywords have a 200 (octal) in the TYPE byte. An unlimited number of keywords can be used.

The only drawback of this fix is the fact that it is not compatible with the standard TPARS. All programs using TPARS must be reassembled and built again.

The fix involves changing the TRAN$ macro and a small patch to the .TPARS routine. You may want to keep them separate from the originals to maintain compatibility.

The patch to TRAN$ can be done by extracting the macro from RSXMAC.SML and using a SLP correction file:

```
  > LBR TRANSORIG.MAC = LB:[1,1]RSXMAC.SML/EX:TRAN$
  > SLP @TRANS.COR
```

TRANS.COR contents:

```
  TRANS.MAC=TRANSORIG.MAC
  -29,37
          .MACRO          $$$TYP
          .BYTE           200
          .ENDM           $$$TYP
          $$$KEY   =      $$$KEY + 1
          $$$FLG   =      $$$FLG ! 1
          .MACRO          $$$EXT
          .WORD           $$$TMP
          .ENDM           $$$EXT
  /
```

The revised module could then replace the original, if desired:

```
          >LBR LB:[1,1]RSXMAC.SML = TRANS.MAC/RP
```

The patch to TPARS changes five words. The simplest way to do this is to use ZAP.

```
  >LBR TPARS.OBJ=LB:[1,1]SYSLIB/EX:.TPARS
  >ZAP
   ZAP>TPARS.OBJ/AB
```

LOCATION	CHANGE	TO
2:202/	042700	16500
2:204/	177700	1
2:206/	006300	240
2:210/	061600	240
2:212/	011000	240
1:16/	146400	146401

The last word changes the version from V0122 to V0122A. The revised module could then replace the original, if desired:

```
          >LBR LB:[1,1]SYSLIB/RP=TPARS
```

CCL Can Help Protect Your System

Eliezer May

Tadiran Electronics Ltd./Systems Division
P.O. BOX 267
Holon, Israel

PROBLEM:

A malicious user who has access to the computer and disks (beyond my control to change this) and gets into the system files after stealing "privilege status" (generally by aborting the startup procedure).

HISTORICAL BACKGROUND TO CCL ON MY SYSTEM

After writing a primitive CCL processor and trying a few CCL's from DECUS RSX tapes, I found the version from James Downward of KMS FUSION, INC. to be excellent. With its help i began buildin a system of ENGLISH, easily understood commands for my users. (Down with non-descipt acronyms followed by esoteric switches!) Next I decided to use the capability of "flying install", run, and remove in order to save pool space for tasks that need not be installed permanently. This worked fine for me but the install failed for my non-priviledged users. To solve this, I installed a simple fix to CCL to turn on the priviledge bit if the command being processed is found in SYSCCL.CCL and to restore the status upon completion. In effect this parallels the use of priviledged tasks by non-priviledged users if such tasks are installed. RSX checks the priviledge at the time of install and not run. Therefore the system manager can control which priviledged tasks are safe for the users to use. Similarly, this allows the system manager control over which commands the user can use in priviledged mode from the extended system commands, i.e., SYSCCL.CCL.

PROTECTING THE SYSTEM

The first thing that must be done is to assign appropriate access permissions to all system files, i.e., no permission to delete or modify. The users may either execute or read as is appropriate. Do not forget files in [0,0].

The next problem is to eliminate his access to priviledge mode whereby he can overcome the file protections. As a further safeguard in case he does somehow find an unguarded terminal in priviledge mode, hide ACNT with a different name somewhere obscure in the system. This is of vital importance. The first thing my malicious user did was to get a listing of all user accounts and passwords. The two methods he used were dumping the file with the account information and ACNT.

The final problem deals with protecting against privileged access at the time of booting the system. I first after making a copy of the system disk (the development of such protections will frequently render the disk unusable and must be restored) VMRed RSX11M to start off TTO as non-priviledged and in a non-privileged UIC. Also redefine CO to the null device. This further protectsagainst an abort attempt during "wake up". The normal processing commands in [1,2]STARTUP are separated into different files with a complex chain of indirect command file chaining between them. The first command in STARTUP initiates a command sequence handled by CCL to prepare a terminal to execute the privileged commands

but block users from entering this terminal. Next we do the normal REDs that RSX would normally do on wakeup. Since CO: is redefined to NL:, the normal messages and REDs disappeared. We also fake the normal print out of these messages by copying a canned message from a file that looks like the normal message. Next in the sequence is the login dialog. We first open a file in another privileged user account by .OPEN command and as a result of the dialog sequence build the sequence of privileged commands that must be executed. The entire sequence on the consol is unprivileged. We now use FRC (see DECUS tapes) to log on a receive-only terminal or some terminal that will execute the privileged commands but can not be intercepted by the user and aborted. Some techniques for such a terminal are: a receive-only terminal (with no keyboard), setting a terminal to a speed lower than 300 baud if the jumper on the DZ's distribution panel for that port is cut, or setting the speed DZ's port speed to 7200 since VT100s do not have a 7200 rate. Next we force a HEL message to the terminal to login a privileged account. This automaically starts its LOGIN.CMD filewhich then starts the command file that was previously built, deltes it, restores its terminal attributes to what they are supposed to be and exits. It is a wonderful experience to see a receive only terminal login to a system and start working on its own initiative. Also hide as much of the commands as possible to the users eyes by either using .ENABLE QUIET and/or putting them in SYSCCL. One may continue ad infinitum complicating the processby different versions of CCL and SYSCCL.CCL, FRCing commands all over the system,and chaining of either tasks (spawn is helpful here) or indirect command files. By the way the .ENABLE QUIET did not work on my RSX 3.2 system but did on 4.0. Also there are tasks around that simply turn off or on the privilege bit for the user. I strongly recommend NOT having such a task installed in the system.

One way of hiding ACNT is renaming it to something like BASLIB.OLB and putting it with the libraries in [1,1] on the system disk. (We don't use BASIC) Pick a name consistant with a possible language but one not in use. This way the file can be inconspicuous in that it is contiguous but not an obvious candidate for a task image search command.

Although this doues not eliminate all threats to system stability if does offer fundamental protection against some of the most dangerous aspects of the system in its most vulnerable state.

An RSX-11M Device Driver Implementing Network Protocols on the DR-11W

D. Burch
V. White

Computing Department
Fermi National Accelerator Laboratory
Batavia, Il. 60510

ABSTRACT

At Fermilab, DR-11Ws have been used as high speed data links to interconnect PDP-11s under both RT-11 and RSX11-M. An imolementation for VAX-11s (under VMS) is planned. Using this hardware, several processors can be used to provide distributed data collection, data monitoring and control, in real-time, for physics experiments.

This paper discusses a device driver which we have implemented under RSX-11M V3.2. This driver allows task-to-task logical connections to be established between a pair of connected PDP-11s. This driver implements several unique software mechanisms including non-standard ASTs, driver maintained queues, and task rundown support. It allows a network operation to be processed with less than 6 ms software overhead.

The reasons for choosing this method of implementation are discussed, and problems solved are described.

Plans for the future development of utility software will be briefly mentioned.

Introduction

The Fermi National Accelerator Laboratory (Fermilab) is a national laboratory dedicated to pure research in the field of high energy physics. Research is performed by a large number of experimental groups each

using a dedicated minicomputer for data acquisition. Experiments at Fermilab are based on the use of a proton accelerator (proton synchrotron) which provides an intense beam of protons for one second out of every ten second accelerator cycle. During this one second period (known as the accelerator "spill"), the minicomputer for each experiment acquires as much data as possible from the experimental apparatus. The data is accumulated in the form of "events", each containing the information recorded by the experimental apparatus for a single interaction between the elementary particle beam and the atoms in a target. Depending on the experiment, the size of an event can vary from one word of data to several thousand words of data. In many experiments, the amount of data which can be accumulated during each spill is limited by the rate at which the computer can transfer the data from the experimental apparatus. Data acquisition rates can run as high as 300,000 words of data per spill, with typical rates being about one quarter of this.

In addition to accumulating data and storing them, normally on magnetic tape, the online computers must monitor the accumulated data and carry out direct checks of the hardware to verify the proper functioning of the entire experimental apparatus.

Fermilab is currently embarked upon a program of improvement in order to meet the future needs of high energy physics. This program involves the construction of a new proton synchrotron that will provide protons with energies of 1 TEV (One Trillion Electron Volts). The principal effects of this program upon the data acquisition environment are the increase of the spill time to twenty seconds out of every sixty, and a marked increase in the complexity of the apparatus of some experiments.

In order to meet the data taking needs of this new era in high energy physics, it became apparent that current data acquisition systems were not adequate. After consideration of many alternatives, it was decided to implement a distributed processing system in which PDP-11s and VAX-11s are connected by a high speed DMA-type communications device. We chose the DR-11W.

The DR-11W is a device capable of transmitting data buffers from one UNIBUS to another in DMA mode. It does so (in non-burst mode) at a rate of three microseconds per word. The peak (burst mode) rate is about two microseconds per word (8,000,000 baud). It is capable of operating only over short (fifty foot) distances, but this was not a limitation in our environment. The DR-11W is low in cost, readily available, DEC supported, and available from a second source.

Link Protocol

In this section we will consider the protocol steps involved in a normal transfer operation of a single block of data (referred to as a "packet") between one and 32,000 words in length. Exceptional cases such as errors, restarts, collisions, and timeouts will not be discussed here. We will note, briefly, the possible exceptions to this protocol which cannot be considered errors.

Our link protocol supports both the transmission of blocks of data and single byte values known as Signals. A signal is transmitted with a shorter protocol exchange, and therefore is suitable for the rapid transfer of simple control data. Data blocks are moved in DMA mode, and protocol exchanges occur in programmed data transfer mode.

The DR-11W is a half-duplex device, and therefore a bid for ownership of the link must occur before a transfer operation can take place. A link operation starts with one side of the link (the Sender) issuing a Request Link. It's partner (the Receiver) responds with a Link Acknowledge.

Next, the Sender transmits a Packet Type Code (PTC) to the Receiver. The Receiver responds to the PTC either with a PTC acknowledge or a PTC NACK. The PTC NACK exists to allow a Receiver to reject certain types of link transactions.

Next, the Sender transmits either a Word Count for the transfer to be performed, or a Signal. A signal can be identified because the sign bit is set and a the upper byte is filled with a particular pattern. The Receiver responds either with a Signal Acknowledge, with A Permitted Word Count, or with a Word Count NACK. The Permitted Word Count is the amount that the receiver actually wishes to receive. By means of this feature, a transmitter can send pieces of a large buffer to a receiver which does not have a buffer large enough to contain the complete message. The Word Count NACK allows a Receiver to reject a transmission which is for some reason not of the correct length. If the receiver sends a permitted word count, it first sets up it's DR-11W for the DMA operation.

Following the receipt of the Permitted Word Count, the Sender sets up its DR-11W device for a DMA, and begins the DMA operation.

After completion of the operation, the Receiver determines whether it has received all of the expected words, and also whether it has any messages waiting for the link, and from these determinations forms an End of Message Status Code. The Receiver then transmits the End Of Message Status Code to the Sender. The capability of a Receiver to request the link at the end of a link operation is important in cases where a Sender is much faster than a Receiver, and could turn around and win the link bid process after each transmission. It also reduces the number of protocol steps during times when the link is seldom idle. The End of Message Status Code completes the link operation. See Figure One for a diagram of this process.

Program-to-Program Protocol

Implied by the Link Protocol is the Program-to-Program or Network Protocol. This system does not address the concept of "route-through". It only provides for communication between a pair of connected machines. To allow all members of a network using this system to communicate with one another without providing application specific routing software, one would have to connect each machine to every other machine in the net.

The Packet Type Code (PTC) identifies a logical link from some procedure to a Receiver. In a multi-tasking system, such as RSX or VMS, this value is associated to a particular task. A "Packet Type Affinity" is then said to exist. By this method, a data block can be sent from a buffer within a task running one machine directly to a buffer in a task running in the receiver machine.

In order to make the operation of a system using the link as insensitive as possible to the type of operating system running in a connected machine, we chose numbers rather than task names to identify logical links. The PTC destination address mechanism provides a logical link in one direction only, that is, from a sender to a receiver. The receiver has no information about the sender of a message other than which LUN the message was received from, and therefore, which physical machine the sender resides in.

For this reason, application code protocols must exist if the receiver of a message must know anything about the message's sender.

Depending on the implementation of this protocol, it is possible for a task using the link to be informed of the nature of a packet before it specifies a buffer for the receipt of the packet. The task then has the option of specifying a sufficiently large buffer, some smaller buffer if wants less than the entire buffer, or rejecting the transaction entirely, which causes the link serving mechanism to issue a Word Count NACK.

The signal mechanism is intended to provide a low overhead mechanism to transfer control information. In the case of RSX-11, the Receiver does not need to issue an I/O operation to receive a signal. In place of an I/O operation, an AST is forced to the task with the signal on the AST stack.

Each Task willing to receive a message must declare which PTCs it intends to receive. Note that one and only one task may establish affinity for a particular PTC at a given time, but as soon as that affinity is removed (either explicitly or through the demise of the task), any other task may obtain an affinity for that PTC.

Note that all data blocks are moved in DMA mode directly from one task's buffer into another task's buffer. No intermediate pools or buffers are involved.

The RSX-11M Device Driver Implementation

The RSX-11M implementation of the above protocols is in the form of a device driver. Tasks using the Communications Driver (CD:), communicate with it via the QIO directive. Currently, this driver has been implemented as either a loadable or resident driver for 18-bit mapped PDP-11s. It is capable of handling communications to any number of DR-11W connected processors, as well as task-to-task communication between tasks running in the machine in which the driver resides.

This latter feature is interesting because it allows all programs in a system to communicate with each other via a standard interface. This allows functions in the distributed processing system to be independent of their position within the system, and allows debugging of software on a single machine. The internal com-

munications option can be generated into a driver which does not support any physical DR-11W units. It can, therefore, run on a 22-bit mapped PDP-11.

In order to create a Packet Type Affinity, a special I/O function is executed which specifies the PTC that the task needs to receive, and an AST address which will be used upon receipt of a signal or when a message arrives with no read queued for that PTC. As described above, this allows a task to accept link transactions without always having an outstanding I/O packet. This allows the task to be checkpointable. Two I/O functions are provided to remove the Packet Type Affinities for a task. One removes a specific Packet Type Affinity, while the other removes all Packet Type Affinities for the issuing task.

I/O functions have been implemented to read packets, write packets, and send signals. The write function specifies a packet address, packet length, and destination PTC as I/O parameters. The unmodified form of the read function is intended to receive a currently pending packet (one for which the task has received an AST). Since there can be one and only one pending packet for each DR-11W unit, the read function specifies only the buffer address and length. Note that until the Receiver issues it's QIO and the transaction completes, no other I/O may occur on that physical unit. Therefore any long series of transactions in this mode will decrease link throughput. An I/O subfunction has been defined that will allow a read operation to be queued to a particular PTC. This read operation will remain queued until a transaction is received for the PTC to which it is assigned. This is the recommended mode when large amounts of data are to be transferred.

When a task receives an AST informing it that a packet is pending for a PTC for which it has declared affinity, but it is unable to process that packet for some reason, it can issue a special I/O function which causes a Word Count NACK to be issued, thereby rejecting that transaction and freeing the link.

At Fermilab, we saw the need for some tasks to periodically gain exclusive use of the DR-11W link, preferably without device driver intervention to avoid the system overhead time of performing a QIO. For this purpose, a subfunction bit has been defined for both the read and write QIO functions that will direct the device driver to "sleep", that is, to give up it's ownership of the DR-11W interrupt vector. The intended use of this function requires that a pair of cooperating tasks exist on either side of the link which is to be put to sleep. One of these tasks initiates the transaction by issuing a write-and-sleep operation to a pre-defined PTC. The other task responds to the receipt of a packet AST for that PTC by issuing a read-and-sleep operation. Upon completion of this transaction, the CD: driver in each machine removes it's interrupt vector and replaces it with the address of the nonsense interrupt handler. This fully interlocks the sleeping process.

When the cooperating tasks have finished their use of the link, they each disconnect from the DR-11W interrupt and issue a special I/O function which restores the driver's interrupt vector and starts the link startup protocol.

While the DR-11W interrupt is removed, link transaction I/O operations are rejected unless they contain a special subfunction bit which tells the driver to allow them to be queued until the interrupt is regained.

7.0 INTERESTING DETAILS OF IMPLEMENTATION

One of the more interesting details of the RSX-11 implementation of this protocol is that it does not depend on the Ancillary Control Processor (ACP) mechanism. Originally, we had intended that the device driver be a much more modest entity, and that an ACP would handle the individual steps of the protocol and maintenance of the PTC tables. When we investigated this path, we found that the overhead time introduced by the ACP scheduling mechanism was longer than our target for total overhead time.

In order to perform the complex protocol and housekeeping chores, the device driver itself had to be considerably more complex than most RSX-11M drivers are (or should be).

The CD: driver handles all of it's own queuing of I/O request packets. Write type I/O operations are queued to the standard I/O queue when needed (when the driver is busy upon receipt of the packet). Read operations are handled quite differently.

When a task creates a Packet Type Affinity, a Packet Descriptor Block (PDB) is allocated in the system pool. This PDB contains the TCB of the owning task, the PTC it refers to, and the AST address specified by the task. The PDB is then linked to the end of a singly

linked list whose head is contained in a Unit Control Block (UCB) extention. Contained within the PDB, and initialized at creation time, is an I/O queue header. When a read operation is received for a particular PTC, that is, with the queue subfunction bit set and the third I/O parameter referring to a PTC for which the issuing task has declared affinity, the I/O packet is queued to this PDB I/O queue. For a picture of these pool structures, see figure two.

One particularly nasty feature of the DR-11W when used as an interprocessor communications device is that interrupts from a connected machine may be lost if the Interrupt Enable (IE) bit in the device's CSR is turned off. This feature dictates some of the more interesting elements of the CD: design.

The CD: driver maintains two independent execution streams. One stream handles the I/O functions that transfer data. It is driven by the completion of I/O operations if the device is busy, and started by either the receipt of a QIO packet or by an unsolicited link transaction. Transfer functions are terminated through the $IODON executive subroutine. The second execution stream handles so-called control or housekeeping functions. These functions are never queued, and are executed immediately upon receipt of the QIO packet which specifies them. Control functions are terminated through the use of the $IOFIN executive subroutine.

Because an interrupt from the DR-11W may occur at any time during the protocol and in any driver state, the management of interrupt and fork-level code is complex. Interrupts can arrive faster than we expect them to, or some error may have occurred in the machine the driver is communicating with, causing a link startup code, or simply garbage, to be received. To handle these situations, the driver follows certain rules such that it's current state is always known while it is executing at below the priority of the device interrupt. At the interrupt entrypoint, a dispatch/action state table is consulted. If any currently executing fork-level code needs to be informed of the result of the interrupt, state flags are set which the fork-level code is required to consult before it terminates it's operations. This function is simplified through the use of central routines which effect the entry to and exit from a particular state.

As was previously mentioned, the CD: informs a task of the receipt of a signal or an unsolicited transaction through the use of an AST. This non-standard

AST block is allocated from pool, and queued to the task using the $QASTT executive subroutine. The AST stack contains the PTC which generated the AST, the unit number of the interrupting unit, and either the word count for the data packet or the signal. Including the unit number and the PTC allows one AST routine to suffice for a task which connects to several units and/or has multiple Packet Type Affinities.

Because the CD: driver maintains non-standard I/O queues, it must intercept and process I/O Kill functions. Any write QIO packets that are on the system I/O queue for the unit are removed by the executive, and then the driver is called to complete the operation by scanning the PDB I/O queues.

When a task exits, it is important that any pool structures which it has created are removed. To allow the CD: driver to perform this function, an RSX-11M I/O function known as IO.CLN is used. In an RSX-11M system, IO.CLN is issued by the I/O rundown service when a task exits with a file open. The intention is that the FILES-11 ACP will intercept the function and close the file. The CD: driver marks the second word of the LUN entry in the header of any task which creates a Packet Type Affinity. This fools the I/O rundown service into believing that a file is open on that LUN, and it sends the driver an IO.CLN I/O request packet. The driver responds to this packet by removing all PDBs owned by that task.

An interesting build-time option of this driver is the trace feature. If the trace feature is present, the driver looks through the Partition Control Block (PCB) chain for a partition called CDTRAC. If this partition is available, some words at the beginning of it are initialized, and the partition is marked as busy so that nobody tries to load anything over it. At key points in the driver's code, a TRACE macro is invoked. This macro creates a call to a tracing subroutine and passes a parameter list to it. The trace subroutine checks a flag in the UCB to see if the trace partition was initialized, and if so it checks a bit in the partition to see if tracing was enabled (the trace enable bit is set by an external task). If tracing was enabled, it moves the current clock tics into the next free position in the trace partition (defined by pointers at the beginning of the partition) followed by the parameter list. A dump program has been written which decodes the data in the trace partition and displays them on a terminal or a lineprinter, and which can enable or disable the trace feature. This allows rapid fault isolation. This code

and the dumping task are sufficiently modular that the writer of any new driver should consider using them.

Timing and Size of Code

The CD: driver may be generated with any combination of several options. It may contain (or not) physical DR-11W support, intraprocessor (internal to a single machine) communication support, and the trace feature. We assembled the driver using the combinations of these options to determine code size. The results were:

1. Physical, Trace, and Intraprocessor - 2859. words.
2. Physical, no Trace, and Intraprocessor - 2011. words.
3. Physical, Trace, no Intraprocessor - 2601. words.
4. Physical, no Trace, no Intraprocessor - 1805. words.
5. No Physical, Trace, and Intraprocessor - 1383. words.
6. No Physical, no Trace, and Intraprocessor - 899. words.

The size of the pool structures needs to be considered when planning a sysgen for a system which will use this driver. The following numbers are for a loadable driver for a non-multi-user system. The Device Control Block (DCB) is fifteen words. The size of the Unit Control Block (UCB) is thirty-three words, and must be repeated for each unit. The size of the Status Control Block (SCB) is twelve words and must be repeated for each unit. Each Packet Descriptor Block (PDB) takes up seven words.

The following average timings were made for communications between an 11/34 and an 11/50;

1. One word packets - 5.7 ms/Transfer.
2. Ten word packets - 5.8 ms/Transfer.
3. 500 word packets - 7.3 ms/Transfer

4. 1000 word packets - 8.8 ms/Transfer.

5. 2000 word packets - 11.9 ms/Transfer.

6. 4000 word packets - 17.9 ms/Transfer.

These timings compare favorably with those for other high speed devices. We timed transfers to an STC-1900 tape drive at 6250 BPI, and found that a 4000 word transfer took 24.3 ms on average. A 4000 word transfer to a TU-10 (800 BPI) took 247.5 ms on average, to no one's surprise.

We ran the intraprocessor version of the driver on a PDP-11/70, and recorded an overhead time of less than 5 ms.

Note that when the Trace feature is in use, the overhead per transfer goes up markedly.

Miscellanious Asides

There are a few things that ought to be noted in passing.

The RT-11 implementation of this protocol is compatible with the RSX-11M implementation, permitting applications under the two operating systems to communicate.

The intraprocessor communications feature uses the $BLXIO executive subroutine to transfer data from one task buffer to another. If this feature is to be used in your system, you should specify the maximum size for the $BLXIO vector in your SYSGEN options generation session.

The link startup protocol depends upon having a source of essentially random numbers to function dependably. We used the location $IDLCT which is a count of idle cycles updated by the rotating data light support routine. Unless you wish to modify the driver, we suggest that you include this option in your system generation. We routinely run our system on 11/34s and 11/60s which do not have data lights, and have experienced no problems with that to date.

We have created a subroutine package, called CDPACK, which provides an operating system independent, FORTRAN-callable interface to the communications services. Currently, versions of this package written for

RT-11 and RSX-11M are completed, and a VMS version will be produced as VMS support is added.

We are currently adapting our Data Aquisition and Analysis systems to use the DR-11W services. Our future projects will probably include remote access to CAMAC device drivers through the use of ACPs and remote servers, File Transfer, and some sort of virtual terminal support.

The current version of the CD: driver, the trace dumping task, and CDPACK will be on the RSX-11 Special Interest Group tape for this DECUS symposium.

Conclusion

The DR-11W-based communications architecture developed at Fermilab is suitable for situations in which rapid data transfer is more important than such features as error detection and retry, although such features could be added with an application layer. This architecture is currently being used to develop applications to support the high energy physics community at Fermilab, and its widespread use at Fermilab is forseen.

REFERENCES

1. P.Heinicke, J.Biel, D.Burch, R.Dosen, M.Pyatetsky, D.Ritchie, V.White, 1982 "High Speed Interprocessor Data Links Using The DR11-W", 1982 Fall DECUS U.S. Symposium, Anaheim, CA.

2. M.Pyatetsky, P.Heinicke, D.Ritchie, V.White, 1982 "Using the DR-11W DMA Device for Interprocessor Communications", 1982 Fall DECUS U.S. Symposium, Anaheim, CA.

Simplified Link Protocol

```
      Sender                                        Receiver
      ------                                        --------
                        Request Link
                ======================
                          Link ACK
                ======================
                 Packet Type Code (PTC)
                ======================
   PTC ACK or PTC NACK
=======================
  Word Count or Signal
=======================
  Permitted Word Count,
  or Signal ACK,
  or Word Count NACK
=======================
  N Data Words (DMA)
-----------------------
  End-of-Message Status
=======================
```

figure 1

RSX-11M V4.0 System Generation

Ralph W. Stammerjohn

Monsanto
800 N. Lindbergh
St. Louis, Missouri 63167

We have recently completed generating RSX-11M V4.0B for the nine Monsanto PDP-11 systems. We set out with three major goals:

1. Minimize the differences between systems as much as possible. We wanted to keep the actual number of copies of various task images, especially privilege tasks, to a minimum.

2. Make the system as friendly to use from the terminal as possible.

3. Maximize the amount of available executive pool. We have historically tried to do more on a RSX-11M system then we should and pool has always been the limiting factor.

In general, we have met all of these goals. This report covers the key actions we took, concentrating on changes to RSX-11M that helped us reach our goals. We also discuss various problems we encountered and our system generation methodology. Specific patch files can

be found at the end of the article.

8.0 MINIMIZING SYSTEM DIFFERENCES

The nine separate systems include PDP-11/23's, 11/34's, 11/44's and 11/70's. All systems have different device configurations and even when they support the same device, the hardware addresses are different. All systems have memory management and floating point units.

We first listed the executive features we required for each system. We then did a dummy system generation and selected the union of all features. The actual differences were minor because we have consistently used the same approaches in each application. We included the new executive commons, pool monitoring and alternate CLI support. Specific features not selected included powerfail recovery, set system time, group global event flags, connect-to-interrupt, disk writecheck, and XDT. The first two are not needed because we have UPS and time-of-day clocks on all systems and the last four features are not used or cost significant pool space.

We also selected the union of all devices between our systems during the dummy generation to see what configuration symbols would be generated.

When we examined the resulting configuration file (RSXMC.MAC), it was clear the only differences were extended memory support (22-bit addressing), the crash dump device, and actual peripherals.

A RSX-11M privilege task uses the symbol definitions in RSX11M.STB to map references in the executive. If all global references used by a privilege task are identical between two executives, the same copy of the privilege task can be used on both systems.

We decided to restructure the executive so only two different versions of privilege tasks would be needed, one for 18-bit systems (11/23, 11/34) and one for 22-bit systems (11/44, 11/70). We would build unique executives for each system. The following changes were made:

1. The crash dump module (CRASH.MAC) conditionally assembles unique code for the specific crash device. Our first thought was to make the length of each code segment the same, thereby making the position of modules following the crash module the same in all systems. This ap-

proach was discarded because it would be difficult to maintain and waste space on some systems.

The approach used was to split the crash module into two files, CRSHP and CRSHC. The first file has the crash dump stack and the crash dump entry points. The module would be the same size no matter what crash device is selected. The second module has the actual crash dump code and contains no global references used by privilege tasks. We placed this module at the end of EXCOM2. A side benefit was an increase in pool because the crash code no longer subtracted from the 20KW kernel executive.

2. All other code in our two executives was the same until we reached the device tables (SYSTB.MAC). We looked at the dummy SYSTB.MAC file that had the union of all devices and checked to see what global references were made by privileged tasks. These turned out to be the console terminal (TTO:), console device (COO:), and several psuedo devices. In addition there are some special structures used by the terminal driver.

   We needed to make the references have the same addresses on all systems. We did this by editing a new module (DEVTB.MAC) from the device database code that had TTO:, COO:, NLO: and the various psuedo devices. Then unique databases for the remaining systems were created as separate files.

3. The only module which follows the device databases is the system initialization code (INITL.MAC). This module does the basic system startup when a virgin image is booted and then links itself into pool. The only global label in this module is at the start of the module ($POOL). This is used in various privilege tasks to find the start of pool.

   We added a new word to the executive common data base (SYSCM.MAC) named $POOLA and modified INITL.MAC to store its starting address in this word. Then all references in privilege tasks to $POOL were changed to use the contents of $POOLA.

These three steps allowed us to build two dummy executives, one for 18-bit systems and one for 22-bit systems, and use the resulting RSX11M.STB files to link two sets of privilege tasks that would work on all systems. We setup RSXMC.18B and RSXMC.22B as the configuration files for both systems and created two separate object libraries. In separate accounts for each actual system we put a RSXMC file that had the specific system name, CPU, and crash device, the system unique device configuration source, and the executive build files. The only unique objects needed for each system were CRSHC (the unique crash device), SYSTB (unique devices), and SYSCM (unique system names).

All of our device drivers are loadable. Device drivers have internal storage based on the number of devices and controllers, but this is typically only one or two words per device. The RSXMC.18B and RSXMC.22B configurations had the union of all devices so we assembled the drivers using these files and generated just two sets of loadable drivers.

The same approach was used with DECnet support. For configuration reasons, Monsanto uses DECnet-11M V2.0. We generated two versions of DECnet that had the union of all network devices used in the system. Each system then as a unique configuration file (CETAB.MAC) that loads only the actual device and sets its vector and CSR address and other unique network parameters. We actually generated four versions of device drivers and DDCMP in order to support whether the system had a KG11 or not.

In summary, we were able to support nine different RSX-11M systems with just two sets of privilege tasks and loadable device drivers. The executive and driver source modules are assembled only two times, once for 18-bit support and once for 22-bit systems. Unique CRSHC, SYSCM, and SYSTB object files are made for each system and specific RSX11M, EXCOM1, and EXCOM2 images built. All privilege tasks and device drivers are linked to the dummy symbol table file for 18-bit and 22-bit systems.

9.0 SYSTEM FRIENDLINESS

System friendliness at Monsanto means letting each programmer use whatever command language they desire and as much shorthand as possible. At the same time, we also have a goal of minimizing pool usage and this means keeping tasks not installed unless they are actually being used.

The KMS Fusion CCL program and INSTALL /CMD are perfect for this purpose. CCL operates as a catch-all task and any commands which MCR cannot directly process are forwarded to CCL. So if a user types MAC A=B and ...MAC is not installed, CCL will get the command line. CCL can then issue an INS $MAC/CMD="MAC A=B" and the assembler will be installed on the fly, do the assembly, and be removed.

The normal command we use at Monsanto to run a task that is not installed is

```
  INS filename/TASK=namtxx/CMD="nam command"
```

where filename is the task image filename, nam is the first three characters of the command verb, txx is the terminal number, and command is the actual command string. For example, our CCL TYPE command would generate the following MCR command when issued from TT11:

```
  INS $PIP/TASK=TYPT11/CMD="TYP TI:=file(s)"
```

This is the same form of command that would be generated by DCL. We chose this command format because we did not want programmers to get confused about what task to abort when changing between DCL and MCR.

However, there is one major problem. RSX-11M limits all MCR command lines to 80 characters. This means the actual command line that can be used with the /CMD switch is 80 minus the overhead of the INS command. The overhead for our shortest form would be the above example or 28 characters. It could be more if we would use an actual filespecification and becomes a problem for commands that normaly have long command lines (MAC, PAT, etc.)

We decided to recode CCL to resolve this and other problems. The new program, CCS, would support the CCL notation with some new extensions but would use TPARS for all parsing, use the new Request and Pass Offspring Information (RPOI) directive, have extended command search algorithms, eliminate parts of CCL we did not use at Monsanto, and solve the problem of the 80 character command line.

1. We used TPARS because it would make it easier to add extensions to CCS command processing in the future and give us better parsing control.

We implement TPARS state tables that handled all of the current CCL syntax and added a %N option. This is the reverse of %Q or execute the command line if the tested parameter is null.

2. The new RPOI directive solved a problem with CCL that existed in V3.2. When a catch-all task is invoked, MCR passes any offspring information (OCB) it has to the catch-all task. But in V3.2 there was no way for the catch-all task to pass the OCB's onto its spawned task (except through executive modifications done by Dan Steinberg to the spawn directive).

   The problem this caused is best illustrated by CCL commands issued from an indirect command file. If CCL exited after spawning MCR with the command line, indirect would continue processing because it thought the command was finished. Also, no exit status would be returned. CCL had to wait until the spawned task would finish and pass the exit status it received along to its parent. CCL's waiting around used unnecessary resources and prevented other CCL commands from being issued when working interactively.

   The RPOI directive solves this problem and we coded CCS to use the directive whenever it constructed the last command in a sequence.

3. CCL would search various places for matches to the command: an internal file, a user file and the system file. We wanted to extend the algorithm to be table driven and add searching by task filename. Our implementation first checks the internal file. Then the command verb is used as the name of a task file and a search is made for the file using a table of dataset descriptors. If the file is found, the command line shown above would be issued with the filename as the command verb. Finally, CCS searches for CCL command files, again described by a table of dataset descriptors, and then searches these files the same way the internal file is searched.

4. CCL had a lot of configuration options we did not use at Monsanto, so while writing CCS, we cleaned up the code and eliminated those features we did not use.

5. The final change is the most critical. When CCS finally constructs a command, it checks the resulting length. If less than 80 characters, CCS uses either the SPWN or RPOI directive to issue the command and either waits or exits.

   But if the command is longer than 80 characters and is in the typical format:

```
    INS filename/TASK=namtnn/sw.../CMD="command"
```

   we removed the /CMD part of the command line and check to see if the actual command line is no longer than 80 characters. If this is true, the following logic was used:

   1. Raise CCS's priority to 255.

   2. Send the command line, without the /CMD part, to MCR to install the task. Wait for INS to finish and take immediate control (because we are the highest priority task in the system).

   3. Now issue either the SPWN or RPOI directive to the task directly and pass the actual command line.

   4. Enter system state and find the Task Control Block for the target task. Set its remove-on-exit bit (T3.REM) and propagate CCS's MCR prompt bit (T3.MCR). Turn CCS's prompt bit off and exit back to task state. This will cause the target task to be removed when it exits and cause a CLI prompt.

   5. Return back to CCS's installed priority.

   The logic works quite well, at the cost of making CCS into a privilege program. We are now able to run our systems with only two installed MCR tasks, INS and HEL. All other tasks are installed only when actually used and there is no command line penalty for the flying install.

There were other actions taken in the area of system friendliness. First, INS has an option in the task build command file to allow non-privilege users to install non-privilege tasks. This is the option statement

```
  GBLPAT  = INSROT:$PRVT:240:240
```

and is normally commented out. We naturally enablec this so CCS would work for flying installs. In addition we added a similar global label, $PRVP, to allow non-privilege users to do privilege installs. This does not effect system security because privilege programs typically do further security checks and let us invoke programs such as MOU and DMO using CCS. Those tasks that do present a security problem, like BOO, we simply hide in an account that non-privilege users cannot access.

Next, the indirect command processor (ICP) had a task build option for a system library account to search if the command file was not found in the current user's account. This feature is enabled by setting the value of the option statement to the binary default account. We used LB:[1,2], done with the following statement in ICPBLD.CMD

```
  GBLDEF  = D$CUIC:402
```

In addition, we patched the search algorithm in ICP to first search SY:[1,2] before the check in the system account. This let separate projects, all working from different virtual disks, to set up their own unique libraries of command files.

With RSX-11M V4.0, we decided to be more aggressive and use more software from the DECUS tapes. We examined all the tapes and compared different versions of various common items. We use the following under RSX-11M V4.0 with no problems:

TECO	V36	Spring 1980	[343,002]	Text editor
DOB	00	Fall 1980	[301,052]	Object disassembler
REI	V01	Fall 1980	[307,022]	Undeleted file
GREP	010102	Fall 1981	[315,100]	Pattern search
RUNOFF	M2.4	Fall 1981	[305,302]	Text formatter
SRD	6.1	Spring 1981	[373,004]	Directories
	plus	Fall 1981	[352,004]	(patches)

Finally, using the version 6.1 of SRD and the new powers of the indirect command processor, we wrote OPS. This command file takes a SRD command line and prototype MCR commands and issues the MCR commands for each occurence of the files found by SRD. This essentially extends the wild-card powers of SRD and PIP to all tasks. For example, the following OPS command does an assembly of all files created on November 11, 1982.

```
  @OPS (*.MAC/DA:11-NOV-82) (MAC $F,$F/-SP/CR=$F)
```

The prototype command use $x for various field substitution and if no prototype command given on the first line, OPS will prompt for any number of commands that will be issued in sequence for each file.

10.0 MAXIMIZING POOL

The limiting factor for the Monsanto systems has always been system pool. Maximizing system pool takes one of two forms:

1. Maximize the total available system pool. RSX-11M's structure requires system pool to be in the first 20KW's of memory, so the maximum size of system pool is 120000(8) minus whatever code resides in this space. Any code removed from the first 20KW's will increase available pool.

2. Minimize the usage of pool. The key to minimizing pool usage is never to tie pool up for inactive users, use as much secondary storage as possible, and keep the size of pool structures to a minimum.

The first steps we took to maximize total available space were obvious: select executive commons, use loadable drivers for all devices, and use the loadable task loader. The next step was to select only the executive features needed for our systems so unused features were not assembled. We did choose the union of all features (see above), however, our most pool-critical systems are also the ones that use the most features.

The only other positive action we did in this area was to move the crash dump code to EXCOM2. This saved around 1400(8) bytes on our largest systems, as well as made it possible to accomplish our first goal. We examined moving other routines to the executive commons

(PARTY, TDSCH, ERROR, PLSUB), but concluded the changes required would not be worth the gain in available pool.

We were more sucessful in minimizing the usage of pool. The Monsanto systems are configured with lots of main memory, ranging from 124KW's on our 18-bit systems to 1088KW's on the 11/70's. Whenever possible, we tried to trade memory for pool:

1. The full-duplex terminal driver (TTDRV) is installed in a 8KW partition. The actual code in the driver is only about 5KW's and the additional space is used for terminal driver pool.

2. CCS helped extensively to reduce the number of installed tasks. We are able to run our systems with only seventeen basic installed tasks and still have the full power's of RSX. We do install SAV and MOU in our VMR command files but these are removed on system startup.

TKTN	Task termination
LDR...	Task loader
PMT...	Pool monitor
MCR...	Command dispatcher
F11ACP	Files-11 ACP
NETACP	DECnet ACP
...INS	Install task
ERRLOG	Error logger
COT...	Console terminal logger
PMD...	Post-mortem dumper
...MCR	MCR command line interpreter
...DCL	DCL command line interpreter
...CA.	Catch-all task (CCS)
...HEL	Login/help task
...AT.	Indirect command processor
PRT...	Print spooler (serial version)
CRF...	Cross referencer

3. F11ACP has internal buffer space that can be expanded in the task build command files. With memory to burn, we decided to expand the large F11ACP (FCPLRG) to 12KW's and make the internal ACP pool ($$AFR1) for File Control Blocks as large as possible. We looked at the normal map and calculated the expansion size for various F11ACP program sections. Unfortunately, this version of F11ACP would trap as soon as mounted. Further investigation showed an undocumented restriction that PSECT $$BUF3 must start before virtual address 160000(8). We redid the

calculations and were able to still add more FCB storage to F11ACP than allowed in the distribution command file.

The distributed version of FCPLRG has the size of $$AFR1 as 3000(8) or 34 FCB's. We are able to increase this to 4400(8) or 52 FCB's before hitting the 160000(8) limit. We still found FCB's being allocated from pool on some systems so we mount different disks with different ACP's.

4. F11ACP has an option to used fixed-size window blocks for open files or attempt to create a window block that exactly maps the entire file (/WIN=FULL). The second option saves pool space when opening small files. If only one retrieval pointer is needed to map the file, then a window block of size 24(8) is allocated instead of the 70(8) when fixed-size windows of 7 are used. Unfortunately, the pathological files that are highly fragmented can cause windows in excess of 1000(8) bytes to be allocated from pool.

   We patched F11ACP to use full mapping instead of fixed-size windows, but to use the volume default window size as the upper limit. We feel this gives us the best of both worlds. Small, existing files only use as much pool as they need. Large, fragmented files do not soak up extremely large segments of pool. Our typical systems have 60 to 70 open files at any point in time so the resulting savings is significant.

5. We found the pool monitor feature of RSX-11M V4.0 to work quite well. A feature we enabled after reading the task build command file was periodic stop task checkpointing (GBLPAT=PMT:CNTRL:160). Every sixty seconds, PMT will force a checkpoint for all stopped, non-ACP tasks. We now see the console logger normally checkpointed has well as many of our own application tasks. Also, the indirect command processors will be checkpointed when they are waiting on a long executions. Any tasks which stop but we do not want checkpointed we simply install with checkpointing disabled.

RSX-11M V4.0, with these additional steps, seems to

have sufficient pool for even our largest systems. Our typical workload is around 25-30 active tasks and free pool from 3500 to 4000 words. We have seen peaks of 45-50 active tasks.

11.0 PROBLEMS AND FEATURES

RSX-11M V4.0B has been fairly problem free. We have only encountered two serious errors, one bad patch, and two 'features' in our conversion from RSX-11M V3.2E.

The first error we encountered were pool link errors that were traced to terminals used DCL. We banned the use of DCL until TSC resolved the problem to a logic error in MCR (MCRDIS). Software Dispatch article 2.2.1.2 (September 1982) fixes the problem. Note, the problem statement is somewhat misleading. Continously striking the PF keys certainly will cause the problem, but any command sent by DCL to MCR and on to the catch-all could corrupt the system.

The other serious bug caused system crashes when a post-mortem dump was made of various tasks. We traced the error to PMD's handling of tasks which map to a 4KW region using APR7, for example a 4KW non-overlaid FCSRES. The error is in PMD's handling of a high virtual address of 177777(8). PMD would increment the value to zero and then subtract the low virtual address to get the length of the region. The resulting bad value then caused bad memory transfers.

Experience and conversations with TSC found the Autopatch B TSHOW.COR patch to DCL is incorrect. Our new version of this file seems to fix the problem and is shown with the other file listings.

The first 'feature' we discovered in RSX-11M V4.0 was slaved terminals no longer had typeahead enabled. All of our actual applications have their own command parsers and run from slave terminals (these were all written before the days of CLI's). Typeahead from these terminals is required to support the block-mode editting done on the terminals. We examined the terminal driver and removed the new feature. Happily we discovered Software Dispatch article 3.1.3.1 (October 1982) had the same patch.

Another 'feature' we discovered concerns the buffer size of the console terminal (CO:). In RSX-11M V3.2, the CO: device buffer size was 255 characters. It appears in the new release that the CO: device takes the buffer size of the terminal used for console logging, typically 80 characters. This broke various parts of our application packages until we found the problem and solved by simply making the buffer size of the console terminal larger.

12.0 SYSTEM GENERATION AND SUPPORT

Monsanto only uses the Digital system generation procedure long enough to generate the template configuration files and system device tables. Once we have this information, we use our own command files to assmble modules and put into the correct system library.

We take a similar approach to task building. We ran SYSGEN3 and generated task build command files for everything. We then edit the command files and overlay descriptors into a standard format and enable or disable various options.

Among the changes we made to the task build command files is to consistently output task images to device TG:, output maps to device LI:, and input all source files from device SY: (except RSX11M.STB and CEX.STB which come from TG:). This lets us easily maintain two virtual disks, one which contains all tasks and maps for 18-bit systems and another for 22-bit systems. We set the state of the checkpoint (/CP/-CP) and floating point (/FP/-FP) switch explicitly and flag all maps for full listings (/MA) and cross reference (/CR). All tasks also have explicit TASK, PRI, UIC, PAR, UNITS, ASG, and STACK options. This lets us do pattern searches to see the various parameter settings for all tasks.

The following is a list of various other actions we did:

1. We are currently using both Fortran-IV-Plus V3.0 and Fortran-77 V4.1 in production and wanted to avoid problems we had in the past with default SYSLIB.OLB searching. If one form of OTS was in SYSLIB, it would be certain that someone using the other compiler would forget and use SYSLIB.OLB as the default. The OTS mismatch would then cause strange problems.

   We decided to put no Fortran OTS (or ANSI FCS support) in SYSLIB. Instead, separate F4PLIB.OLB and F77LIB.OLB OTS-only libraries

were created. We also support SYSF4P.OLB and SYSF77.OLB as default libraries. This forced programmers to explicitly name the OTS they used.

2. We have many application programs that link to the RSX-11M V3.2 FCSRES resident library. In order to avoid rebuilding all these programs, we name the RSX-11M V4.0 resident library DECRES and support both on all systems. We will eventually migrate all FCSRES support to DECRES.

3. Our systems have memory to burn, so we tried to find all tasks which benefit from additional dynamic memory and task built them as 28KW tasks (plus DECRES). This increases system throughput and helps the tasks show up on a 1088KW RMDEMO display. The tasks that fall into this category include CMP, CRF, DSC, FTB, F4P, F77, LBR, MAC, PIP, TEC, TKB, and VFY.

4. We examined the priority of all tasks and changed to better meet our needs. The basic question we asked as if these two tasks both want to run at the same time, which one should. We ended up setting the following priority classes:

   1. Priority 80

      All MCR privilege tasks except INS (ACS, BOO, BRO, BYE, DMO, ELI, HEL, INI, LOA, MOU, PMD, SAV, UFD, and UNL).

   2. Priority 70

      All CLI command processors and video editors (DCL, CCS, EDT).

   3. Priority 60

      The indirect command processor, other editors, and file utilities (AT., TEC, PIP, and SRD).

   4. Priority 50

      All of the other tasks with the exception of those at 40.

   5. Priority 40

Background tasks (CRF) and backup/disk utilities (BAD, BRU, DSC, FMT). The later are normally run at standalone times and when used in production really consume system resources.

There is also a category of system tasks that we put in individual priorities, starting at 245. These task are ranked by descreasing importance, separated by units of 5. This allows us to put our real-time application tasks at exactly the right order with regard to the rest of the system. The current ordering for these tasks is TKTN, LDR, PMT, COL (SPM data collection), RMD, MCR, F11ACP, NETACP, INS, ERRLOG, COT, MTAACP, PMD, and SHF.

13.0 PATCHES AND FILES

The following has many of the files discussed in the previous sections. The source patch files are in SLP autolocator format and the object patches in PAT form.

13.1 Crash Module

The following SLP correction file splits CRASH.MAC into two files, CRSHP.MAC and CRSHC.MAC.

```
CRSHP.MAC/AU:72./-BF=CRASH.MAC

-/.TITLE  CRASH/,.
  .TITLE  CRSHP
-/; MODIFIED BY:/
;
; R. STAMERJOHN   19-JUL-82
;
;         RWS001 -- SPLIT CRASH INTO POOL AND CODE MO-
DULES
;
; R. STAMERJOHN   07-SEP-82
;
;         RWS011 -- MAKE SURE MEMORY MANAGEMENT IS ON
BEFORE JMP
;
%
-/$CRUST::/,/$CRPBF::/,/;RWS001/
```

```
$CRUST::.WORD     0                 ; USER PS IS STORED
HERE

  .IF DF  D$$PARM$$MGE

$CRAR5::.WORD     0                 ; SAVED KERNAL APR 5
VALUE

  .ENDC

                              ; *** THE FOLLOWING MUST BE
ADJACENT
$CRPBF::.BLKW     4                 ;STACK AREA FOR SU-
BROUTINE CALLS
-/$CRSBN::/,/$CRSCS::/,/;RWS001/
-/$CRALT::/,/$BTSTP::/,/;RWS001/
$CRALT::

  .IFT

  .IF DF  D$$PARM$$MGE

  MOV     KISAR5,$CRAR5   ;SAVE OLD MAPPING
  MOV     $XCOM2,KISAR5   ;MAP INTO EXCOM2
-,,/;RWS011/
  BIS       1,@ SR0        ;MAKE SURE MEMORY MANAGEMENT
IS ON
-,,/;RWS001/

  .ENDC

  JMP     $CRSHC          ;JUMP TO CRASH CODE

  .IFF

  HALT

-.,/BR    .-2/,/;RWS001/
-.,.+3,/;RWS001/
/
CRSHC.MAC/AU:72./-BF=CRASH.MAC

-/.TITLE  CRASH/,.
  .TITLE  CRSHC
-/; MODIFIED BY:/
;
; R. STAMERJOHN   19-JUL-82
;
;         RWS001 -- SPLIT CRASH INTO POOL AND CODE MO-
DULES
;
%
```

```
-/$CRUPC::/,/$CRUST::/,/;RWS001/
-/; *** THE FOLLOWING/,/; *** ABOVE/,/;RWS001/
  .IF DF  C$$CDA
-/;+/,/$TRINT::/,/;RWS001/
-.,.,/;RWS001/
-/$CRASH::/,/$CRALT::/,/;RWS001/
$CRSHC::                      ; CRASH DUMP ROUTINE

  .IF DF  C$$RSH

-/ UBMPR/,.,/;RWS001/

  .IF DF  D$$PARM$$MGE

  MOV     $CRAR5,24(SP)    ;;;STORE SAVE KERNEL APR 5

  .ENDC

  MOV      UBMPR,R0        ;;;GET ADDRESS OF FIRST UMR
/
```

13.2 $POOLA Support

The following SLP correction files add the new global variable $POOLA to SYSCM, modify INITL to intialize this variable to start of pool, and change all uses of label $POOL to reference the contents of $POOLA. Note, ST20V and MCRDEF are MCR modules and RMDDEF, MDINIT, THF11F, and THPAGE are RMDEMO files.

```
SYSCM.MAC/AU:72./-BF=SYSCM.MAC

-/; MODIFIED BY:/
;
; R. STAMERJOHN   19-JUL-82
;
;        RWS002 -- ADD $POOLA WORD, HOLDS TOP OF POOL.
;
%
-/$CMFIN::/,.,/;RWS002/
$POOLA::.WORD     0               ;TOP OF POOL ($POOL)
$CMFIN::                      ;END OF SYSCM AREA FOR CDA
/

INITL.MAC/AU:72./-BF=INITL.MAC

-/; MODIFIED BY:/
;
; R. STAMERJOHN   19-JUL-82
```

```
;
;        RWS002 -- STORE TOP OF POOL IN $POOLA
;
%
-/BIC     $CRAVL-2/,,/;RWS002/
  MOV     R0,$POOLA                ;STORE TOP OF POOL
/

MCRDEF.MAC/AU:72./-BF=MCRDEF.MAC

-/; VERSION:/
;
; R. STAMERJOHN   19-JUL-82
;
;        RWS002 -- REFERENCE $POOLA INSTEAD OF $POOL
;
%
-/$POOL/,.,/;RWS002/
  .GLOBL  $POOLA
/

ST20V.MAC/AU:72./-BF=ST20V.MAC

-/; MODIFIED BY:/
;
; R. STAMERJOHN   19-JUL-82
;
;        RWS002 -- STORE TOP OF POOL IN $POOLA
;
%
-/SUB     $POOL/,.,/;RWS002/
  SUB     $POOLA,R3        ;   CURRENT AS TO BOUNDARY
VALUE
/

RMDDEF.MAC/AU:72./-BF=RMDDEF.MAC

-/; MODIFICATIONS:/
;
; R. STAMERJOHN   19-JUL-82
;
;        RWS002 -- USE $POOLA TO GET START OF POOL
;
%
-/$POOL/,.,/;RWS002/
  .GLOBL  $POOLA
/

MDINIT.MAC/AU:72./-BF=MDINIT.MAC

-/; MODIFICATIONS:/
;
```

```
; R. STAMERJOHN   19-JUL-82
;
;         RWS002 -- USE $POOLA TO GET START OF POOL
;
%
-/MOV      $POOL/,.,/;RWS002/
  MOV      $POOLA,R1                  ;GET START OF POOL
  ASR      R1                         ; IN 32W BLOCKS
  ASR      R1                         ; ...
  ASR      R1                         ; ...
  ASR      R1                         ; ...
  ASR      R1                         ; ...
  ASR      R1                         ; ...
  BIC       1777,R1                   ;CLEAR ANY GARBAGE
/

THF11L.MAC/AU:72./-BF=THF11L.MAC

-/; DATE:/
;
; R. STAMERJOHN   19-JUL-82
;
;         RWS002 -- USE $POOLA TO GET START OF POOL
;
%
-/CMP      R2, $POOL/,.,/;RWS002/
  CMP      R2,$POOLA       ;; BELOW POOL?
/

THPAGE.MAC/AU:72./-BF=THPAGE.MAC

-/; DATE:/
;
; R. STAMERJOHN   19-JUL-82
;
;         RWS002 -- USE $POOLA TO GET START OF POOL
;
%
-/CMP      R0, $POOL-1/,.+1,/;RWS002/
CHKADR:    CMP      R0,$POOLA
  BLO      10$
/
```

13.3 INS Support for Non-Privilege Installs

The following SLP correction file adds a new global label, $PRVP, to INSTALL. This allows the error checking for non-privilege users installing privilege tasks to be disabled in the task build command file.

```
INSLB.MAC/AU:72./-BF=INSLB.MAC

-/; MODIFIED BY:/
;
; R. STAMERJOHN   19-AUG-82
;
;         RWS007 -- ADD GLOBAL LABEL TO PATCH OUT PRIVI-
LEGE INSTALL
;                 CHECK FOR NON-PRIVILEGE USER
;
%
-/4912$:/
-/INSL21/,.,/;RWS007/
$PRVP::
  JMP      INSL21           ; NO
/
```

13.4 Indirect Command File Lookup

The following SLP patch adds SY:[1,2] to Indirect MCR's search path for command files. This also requires the task build value of D$CUIC to be set to 402(8).

```
ICPATS.MAC/AU:72./-BF=ICPATS.MAC

-/; MODIFED BY:/
;
; R. STAMERJOHN   10-AUG-82
;
;         RWS006 -- SEARCH SY:[1,2], THEN LB:[1,2] FOR
COMMAND FILES
;
%
-/BEQ      16$/,.,/;RWS006/
  BLE      16$              ; BR IF NOT, TRY SY:[1,2],
THEN LB:[1,2]
-/16$:/,,/;RWS006/
  TST      R4               ; IS THIS SECOND PASS
  BEQ      17$              ; BR IF NO, TRY SY:[1,2]
  MOV      CMBPTR,R0        ; SET PREVIOUS COMMAND
  MOV       "@L,(R0)+       ; SET FOR @LB:
  MOV       "B:,(R0)+       ;  ...
  MOV       1,R4            ; FLAG NO MORE PASSES
  BR       10$              ; AND RETRY OPEN

17$:
-/ "@L/,/ "B:/,/;RWS006/
  MOV       "@S,(R0)+       ; SET FOR @SY:
  MOV       "Y:,(R0)+       ;  ...
```

```
-/MOV     SP,R4/,.,/;RWS006/
  MOV      -1,R4           ; SET SWITCH FOR THIRD PASS
/
```

13.5 OPS.CMD

The following is the command file we use for wild-card support for all utilities.

```
  .ENABLE SUBSTITUTION
  .ENABLE GLOBAL
  .;
  .; OPS - Select files and perform operation.
  .;
  .;      @OPS (SRD selection) (command string)
  .;
  .PARSE COMMAN "()()" DUMMY SRD DUMMY CMD1
  .;
  .; If no command string specified, loop and read
strings.
  .;
  .IF CMD1 <> "" .GOTO SORT
          .SETN CMD 0
.CMDS:            .INC  CMD
          .ASKS XXX Command string  'CMD'
          .IF XXX = "" .GOTO SORT
          .SETS CMD'CMD' XXX
          .GOTO CMDS
  .;
  .; Make SRD selection to temporary file and open for
read.
  .;
.SORT:    SRD SRD.TMP='SRD'/LI
  .OPENR SRD.TMP
  .;
  .; Read SRD file and select filenames.  Parse  so  $N
is entire
  .; filename, $D is device, $U is UIC ($G is group, $O
is owner),
  .; $F is filename, $T is filetype, and $V is version.
  .;
.LOOP:    .READ LINE
  .IFT <EOF> .GOTO FINI
  .;
  .SETS TEST LINE[1:3]
  .IF TEST <> " **" .GOTO FILE
          .PARSE LINE "  [,]" DUMMY DUMMY $D $G $O DUMMY
          .SETS  $U "["+$G+","+$O+"]"
          .GOTO  LOOP
```

```
  .;
.FILE:    .SETS TEST LINE[20:20]
  .IF TEST <> ";"   .GOTO LOOP
          .PARSE LINE "  .; " DUMMY DUMMY $F $T $V DUMMY
          .SETS  $T "."+$T
          .SETS  $V ";"+$V
          .SETS  $N $D+$U+$F+$T+$V
  .;
  .; Enter loop and execute commands for each file.
  .;
  .SETN CMD 0
.CHKS:    .INC  CMD
  .IFNDF CMD'CMD' .GOTO LOOP
          .TEST CMD'CMD'
          .SETN SIZE <STRLEN>
          .SETN CHAR 0
          .SETS OUT  ""
          .;
.CONT:            .INC CHAR
          .IF  CHAR > SIZE .GOTO EXEC
                  .SETS TEST CMD'CMD'['CHAR':'CHAR']
                  .IF TEST = "$" .GOTO SUBS
                          .SETS OUT OUT+TEST
                          .GOTO CONT
                          .;
.SUBS:                            .INC CHAR
                          .SETS TEST
CMD'CMD'['CHAR':'CHAR']
                          .SETS OUT OUT+$'TEST'
                          .GOTO CONT
                  .;
.EXEC:            'OUT'
          .GOTO CHKS
  .;
.FINI:    PIP SRD.TMP;*/DE/NM
```

13.6 F11ACP Windows

The following PAT file allows F11ACP to use exact file window mapping with an upper limit of the volume window size. The original module checksum is 70061(8) and the INWIN.POB file checksum is 16421(8).

```
  .TITLE  INWIN
  .IDENT  /M0226/
;
;         RWS015 - USE VOLUME DEFAULT AS UPPER LIMIT TO
TRY FULL MAPPING.
;
```

```
  .PSECT  INWIN
.PURE.    = .

. = .PURE.+24
  MOVB    V.WISZ(R4),R1            ;GET THE VOLUME DE-
FAULT
  NOP                              ; FALL INTO FULL LOGIC

. = .PURE.+52
  CALL    PAT001

  .PSECT  PAT001

.PURE.    = .

PAT001:   MOV     R1,R3                    ;COPY RETRIEV-
AL POINTER SIZE
  BGT     10$                      ; IF GT - USE AS UPPER
LIMIT
  MOV      <<512.-W.RTRV>/6.>,R3  ;SET MAXIMUM RETRIEVAL
SIZE
10$:      RETURN                           ; AND CONTINUE

  .END
```

13.7 PMD Correction

The following PAT correction file fixes the problem in PMD with 4KW APR7 windows. The orginal object file checksum is 37060(8) and the PMD.POB file checksum is 4122(8).

```
  .TITLE  PMD
  .IDENT  /07.2/
;
; MODIFICATION:
;
; RWS012 -- FIX DUMMPING OF 4KW APR7 WINDOW
;

  .PSECT
.BLK. = .

. = .BLK.+1776
  NOP

  .END
```

13.8 TSHOW Correction

The following is a corrected version of the TSHOW.COR file on the Autopatch B kit.

```
[23,10]TSHOW.MAC;2/AU/-BF=[23,10]TSHOW.MAC;1

-2,2
  .IDENT  /01.01/
-20,20
; LHS032  CORRECTED SHO ACC SYNTAX
%
-36,37,/;LHS032/
  .IF  DF  R$$MPL
  OR      <GNITNUOCCA              BST=B1>
  OR.     <'CLQ                    BST=B2>
  .IFF
  OR      <'CLQ                    BST=B2>
  .ENDC
-55,71,/;LHS032/
  .IF  DF  R$$MPL
GNITNUOCCA:
  AND     <<'SHO ACC >> <MAP=1 OPT=T>
  END
  .ENDC
/
```

DECUS

DIGITAL EQUIPMENT COMPUTER USERS SOCIETY
ONE IRON WAY, MRO2-1/C11
MARLBORO, MASSACHUSETTS 01752

BULK RATE
U.S. POSTAGE
PAID
PERMIT NO. 129
NORTHBORO, MA
01532

MOVING OR REPLACING A DELEGATE?

Please notify us immediately to guarantee continuing receipt of DECUS literature. Allow up to six weeks for change to take effect.

( ) Change of Address
( ) Delegate Replacement

DECUS Membership No.: ____________________

Name: ______________________________

Company: ____________________________

Address: _____________________________

____

State/Country: ________________________

Zip/Postal Code: ______________________

Mail to: DECUS - ATT: Membership
One Iron Way, MRO2-1/C11
Marlboro, Massachusetts 01752 USA

Affix mailing label here. If label is not available, print old address here. Include name of installation, company, university, etc.