THE mini-tasker

DECUS
RT-11 SIG NEWSLETTER

March 1984 Volume 10, Number 2

SJ RTMON RMON FILEX CSI ODT

PIP LD SYSMAC FB

DIR KMON QUEMAN

QUEUE TECO

PAT

RT-11 SIG DECUS U.S. CHAPTER

DUP

VM

K52

KED BINCOM

LIBR DUMP BUP

SIPP

LINK

SRCCOM RESORC FORMAT

SLP

TTYSET IND

MACRO XM JSW HELP

Printed in the U.S.A.

THE mini-tasker

DECUS
RT-11 SIG NEWSLETTER

MARCH 1984 VOLUME 10, NUMBER 2

Contributions to the newsletter should be sent to:

Ken Demers
MS-48
United Technologies Res. Ctr.
East Hartford, Ct.
06108
(203) 727-7139 or 7240

Other communications can be sent to:

John T. Rasted
JTR Associates
58 Rasted Lane
Meriden, Ct.
06450
(203) 634-1632

or

RT-11 SIG
C/O DECUS
One Iron Way
MR2-3/E55
Marlboro, Ma.
01752
(617) 467-4141

TABLE OF CONTENTS

RT-11 SIG CONTACTS

JOHN T. RASTED JTR Associates 58 Rasted Lane Meriden, CT 06450 (203) 634-1632	RT-11 SIG Chairman
KEN DEMERS MS-48 United Tech. Research Ctr. Silver Lane East Hartford, CT 06108 (203) 727-7139 or 7240	Newsletter Editor DECNET Contact
RALSTON BARNARD Division 2565A Sandia Laboratories Albuquerque, NM 87185 (505) 844-5115	RT-11 Handout Editor Tape Copy Generation Contact
MARK BARTELT HSC Research Development Corp. 555 University Ave. Toronto, Ontario Canada M5G 1X8 (416) 597-1500 Ext 4588 or 598-5982	Structured Languages Contact
DOUG BOHRER Bohrer & Company 903 Ridge Road, Suite 3 Wilmette, IL 60091 (312) 251-9449	APL Contact
NICK BOURGEOIS / 9238 Sandia Laboratories P.O.Box 5800 Albuquerque, NM 87185 (505) 844-8088	TSX Contact MACRO Contact
JOHN CROWELL CROW4ELL Ltd. 145 Andanada Los Alamos, NM 87544 (505) 662-3893	TECO Contact Product Planning Contact
BILL LEROY Information Resources, Inc. 470 E. Paces Ferry Rd. N.E. P.O. Box 52661 Atlanta, GA 30355 (404) 231-3117	COBOL Contact Wish List Contact

CARL LOWENSTEIN
Univ. of Calif. San Diego
Marine Physical Lab. of the
Scripps Inst. of Oceanography
San Diego, CA 92152
(619) 294-3678

RT-11 Hardware Contact

JACK PETERSON
Horizon Data Systems
1901 Wildflower Terrace
Richmond, VA 23233
(804) 740-9244

C Contact

SUSAN S. RASTED
Software Dynamics Inc.
1000 Yale Ave.
Wallingford, CT 06492
(203) 265-2226

FMS-11 Contact

NED RHODES
E-Systems
Melpar Division
7700 Arlington Boulevard
Falls Church, VA 22046
(703) 560-5000

Symposia Coordinator

TOM SHINAL
General Scientific Corp.
1684 East Gude Drive
Rockville, MD 20850
(301) 340-2773

RT DECUS Library
Contact
Tape Copy Distribution

ED STEVENS
E M D A Inc.
111 South Hudson St. #B
Pasadena, CA 91101
(213) 795-5991

BASIC Contact

J. W. TIPPIE
Kinetic Systems Inc.
11 Mary Knoll Drive
Lockport, IL 60441
(815) 838-0005

CAMAC Contact

RON TRELLUE
Division 7523
Sandia Laboratories
Albuquerque, NM 87185
(505) 844-0955

FORTRAN Contact
RT-11 LUG Contact

FROM THE EDITOR

I attended the DECUS Publications Committee meeting which was held February 6th and 7th. This annual meeting is scheduled to allow the newsletter editors the opportunity to resolve any current problems concerning the newsletters and to have the committee formulate long range plans for SIG publications. Each editor attemps to represent fairly, his or her respective SIG, based on feedback received at each symposium and received individually from members throughout the year. The issues discussed, that are of significance to the RT-11 SIG are:

1. changing the current newsletter pricing structure to ensure that the price you pay for a newsletter reflects the actual cost of printing and distributing the newsletter you receive

2. the possibility of altering the current groupings of multiple SIGs within a single newsletter

3. improving the turn around time between symposium and scribed material appearing in the newsletters

4. examining the possibility of changing the current copyright statement that appears in each newsletter

In addition, Shelli Kiesling of the DECUS staff made a presentation that explained the process that each newsletter must go through once the DECUS office receives it from a newsletter editor. Shelli also presented various tips that the newsletter editors should follow in order to avoid putting uneccessary delays in the process. I have included a diagram that Shelli gave us that illustrates this process. It should be noted that Shelli is the person behind the scenes that performs the task of getting all of the newsletters into a form that is ready to be sent to the printer. We should all thank her for a job well done.

Last year the subscription fee for each newsletter was the same. This resulted in a newsletter that had two issues costing the same as a newsletter that had twelve issues. Thus, the subscribers to a newsletter offered by the smaller SIGs were subsidizing the subscribers of the larger SIG's newsletters.

The proposed solution is to have each SIG's newsletter subscription fee reflect the newsletter's costs to publish which are determined by 1. page count per issue 2. number of issues per year and 3. number of yearly subscribers. This will allow you to pay a fair price for the information you receive. Under this format, next year's subscription to the 'Mini-Tasker' will cost approximately $14.00 . This figure is based on publishing five issues of approximately sixty pages to the approximately 3,000 current subscribers. The range of projected costs for other newsletters were a low of $5.00 (two issues a year) and a high of approximately $36.00 (twelve issues a year). It was also recommended that subscribers ordering all of the newsletters will be offered a discount (the amount has not been determined yet).

Item 2 does not affect the 'Mini-Tasker'. It is possible that the Basic SIG's newsletter will not be published individually this year and that the RSX and IAS SIG's newsletters will be published individually this year.

There was much discussing among the editors concerning the delay in getting the scribe material from the St. Louis Symposium into our respective newsletters. This was a result in logistical problems related to personnel changes in the scribe distribution chain. (This is a typical problem since we are a volunteer organization). However, the editors will be able to speed up the scribe distribution process in the future since we will be utilizing the DECUS Vax computer to aid in the scribe distribution process.

Item 4 is of concern to those publication committee members that feel that the current copyright statement which appears on newsletters is too restrictive. The committee decided to examine the current policy in more depth to ensure that if any changes are made, they are in the best interest of the DECUS society as a whole.

As a unit, the publications committee is striving to bring SIG members the highest quality newsletters at the lowest and fairest prices. It should be noted that even with the initiation of the subscription fee last year, overall the publication of newsletters lost money. But, experience is a great teacher. With the new pricing schedule and a year's experience, we intend to break even as a non-profit organization should.

As always, your comments concerning the publications meeting, the "Mini-Tasker", or the RT-11 SIG are encouraged and welcomed.

Sincerely,

Ken

USER INPUT

A PHILOSOPHICAL ASIDE

Ned W. Rhodes

In doing some investigations concerning the use of Concurrent Pascal for an application, I came across a list of guidelines pertaining to scheduling algorithms. After reading them, I discovered that one of the principles really has application to the general area of computing and software design. The article "Monitors: An Operating System Structuring Concept" was authored by C.A.R. Hoare and was published in the Communications of the ACM (Copyright 1974, Association for Computing Machinery, Inc., October 1974). The principle that I found interesting was:

"Do not seek to present the user with a virtual machine which is better than the actual hardware; merely seek to pass on the speed, size and flat unopiniated structure of a simple hardware design."

I am sure that it has been said before and in different words, but I found the words interesting. Many times I have attempted to do emulations of specific, dedicated hardware on a general purpose computer, only to find that the emulation doesn't run nearly as fast or efficiently as the thing being emulated. Examples of this would be foreign terminal emulation or executing CPM code on a PDP-11. I have had to take a step back, and realize that I can choose efficient algorithms, but that I can't make the hardware any faster than it already is.

Another point that I found interesting, was Hoare's discussion on system loading and thrashing. He said "avoid fixed priorities; instead, try to ensure that every program in the system makes reasonably steady progress. In particular, avoid indefinite overtaking." I find that this is an interesting thought also. I know that in the design of real-time systems, I have often wanted to have a complicated scheduling scenario, where one program would interrupt another, while still another program could interrupt all the others and soon we would find that the system was busy interrupting itself and not getting the overall job done. I have come to believe that in an event driven system, it is better to let certain tasks run to completion rather than try to build in a complicated scheduling algorithm; it is certainly easier. The easiest way I have found to do this is to make the system queue driven, and have the tasks in the system work from the queue. Then, let each task complete its processing on an element from the queue, either by completely finishing with the queue element or by inserting information on another queue for further processing. This ensures that tasks will make reasonable progress and that you can pass on the speed of the processor in a simple manner.

Dear Mr. Demers,

Enclosed you find two product descriptions, the source of a utility program FF and the subroutine INCLUD together with an example data file. FF sends a FormFeed and a header line with date and time to LP:. It is very handy to separate listings (with the command .FF, if FF.SAV resides on SY:). Of course it could have been much more easily programmed in FORTRAN, but we had the MACRO code already available in another package. The advantage of this MACRO version is it's very small size in memory and on disk (3 blocks).

The subroutine INCLUD is intended to be used for control of programs by data in a parameter file (see enclosed example). Several data types are specified explicitly in the parameter file. Further, in the file, comments may be added at several places, so it's self documenting. Subroutines called by INCLUD are not enclosed as these are part of a large package. The intention here was mainly to present the idea. We use it already a long time to control programs without having to rebuild programs or having excessive terminal input.

Suggestions for improvements or enhancements from other RT-11 users are of course very welcome!

Yours sincerely,

[illegible]

H. Haenen
Dept. Clin. Neurology AZG
P.O. Box 30.001
9700 RB GRONINGEN / The Netherlands

RT-11 DISK DATA CACHE

Purpose: Speedup of disk I/O in a transparent way by caching technique

Characteristics:

+ Considerable improvement of system response, virtually eliminates USR and KMON swapping
+ Applications run much faster as far as disk I/O is concerned
+ After startup caching is completely transparent
+ Requires only about 200 words of resident cache code in low memory. This cache code resides in VM: handler
+ A must for floppy and TU58 users due to dramatic performance improvement.

Principle: Direct Mapping (proved to be superior to Look Ahead and Least Recently Used algorithms): a fixed disk area is mapped to a file(the cache file) on VM: (the virtual memory handler). Space on VM: not occupied by the cache-files remains available for normal use. Up to 10 disk area's may be defined on several units. A disk area may be the directory (in many situations a very efficient cache area), a file or a part of a file.

Failsafe: Uses the Write-Through principle. A read to a cached disk area is served with data from the cache file on VM:. However, a write updates the cache data as well as the disk data. So, even when a disk directory is cached, the disk is not corrupted after a system crash.

Write cache: The Write-Trough principle can be disabled explicitly for specific disk area's which contain temporary data. For example consider the SWAP.SYS system file. The data in this file have only a meaning during runtime. I/O to disk area's cached in this way write to/read from only the VM: cache file.

Software:

+ Code to update VM: handler (V4 or V5) for caching.
+ CACHE, a utility program for starting/stopping caching and testing cache contents
+ CSHOW, a utility program to be run at any time during caching to show (print) cache setup and performance

Required:

+ RT-11 V4 or V5
+ Extended memory. The more memory, the more disk data can be cached. However, memory not used for caching remains available for data storage. A total of 128 Kw. memory in a system may cache up to 376. disk blocks. VM: RT-11 V5 supports upto 2 Mw on Q-bus.
+ Person installing the package should be familiar with system generation (it may be necessary to assemble, modifie and install a handler)

RT-11 MULTIPROCESSOR AND DATA COMMUNICATION PACKAGE

Purpose:	+ Data Communication between RT-11 systems. + Transparent use of remote devices(disks, logical disks lineprinter, Magtape, etc.) with dynamical setting of read/write access protection. + Connects memory-only system to remote system with disk. Memory-only boots from remote disk. + Extends single user RT-11 to a multiprocessor, MULTI-USER environment.

Hardware supported:	DL(V)-11 (DEC, serial)
	DR(V)-11 C/(00) (DEC, parallel)
	WB(V)-11 (Hammond Software, fast serial, opto-coupling)
	Qnector (Westvries Systems B.V., fast DMA interface)

Principle:	At the remote side a service job (file server) runs which acknowledges requests from a data communication handler. Communication is based on a slightly modified Radial Serial Protocol (RSP). Basically bytes, words or blocks of data are transferred, depending on the type of interface (serial, parallel or DMA). Error checking is done on the data transferred. The data communication handler is bootable. Additional (Pseudo-) handlers realize the transparent use of remote devices as disks, lineprinters, Magtape.
Software:	+ Data Communication Handlers. Size about 700 words, depending on options selected (e.g. time-out). + Pseudo-handlers, emulate remote devices. Size about 30 to 60 words, Magtape handler is 180 words. + Service jobs. Minimal size is 1300 words for a job with one 256 word buffer. Additional buffers may be selected at assembly time. Special function and special directory device (e.g. Magtape) support adds 300 words to the size. Jobs run in a FB system as Foreground or System job. Each job can simultaneously access up to 16 devices (could be coded for up to 256 devices). + Job Show utility for printout of nr. transfers done, nr. errors, nr. time-outs etc., read or write access to each device allocated by a selected service job may be changed. + Several utilities for message transfer ("mail"), stopping jobs, print-out of data communication packeds, testing. + Bootstrap programs for PROM or toggle-in. Secundairy bootstrap programs with password control.
Required:	+ RT-11 V4 or V5 + Person installing the package should be familiar with system generation (it may be necessary to assemble, modifie and install a handler)

```
        .TITLE  FF
; FF.MAC program sends FormFeed and header line with time and date to LP:
; H.H. nov 83

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++;

        .MCALL .EXIT,.PRINT,.LOOKUP,.READW,.WRITW,.CLOSE,.SETTOP
        .MCALL .DATE,.DSTAT,.FETCH,.RELEAS
        .ENABL LC
        .GLOBL  TIME
START::
        MOV     @#50,R5         ;Load R5 with high program limit.
        .DSTAT #STAT,#FILO
        TST     STAT+4          ;Handler already loaded?
        BNE     LOADED
        MOV     STAT+2,R0       ;Handler size!
        ADD     R5,R0           ;New high limit needed!
        BCS     MEMOVR
        MOV     R0,-(SP)
        .SETTOP R0              ;Do limit request.
        BCC     1$
        TST     (SP)+
        .PRINT #SETERR
        .EXIT
1$:     CMP     R0,(SP)+        ;Enough memory?
        BCS     MEMOVR
        .FETCH R5,#FILO
        BCC     LOADED
        .PRINT #NOINST
        .EXIT
MEMOVR: .PRINT #NOMEM
        .EXIT
;
;  Open channel to output device
;
LOADED: .LOOKUP #AREA,#0,#FILO
        BCS     LFAILO
        JMP     OK
;
; Error returns:
;
LFAILO: .PRINT  #NOLKO
        .EXIT
;------------------------------------
FILO:   .RAD50  /LP /
        .WORD   0,0,0
;------------------------------------
STAT:   .BLKW   4
NOINST: .ASCIZ  /?FF-F-LP: not in SYSTEM/
NOMEM:  .ASCIZ  /?FF-F-Not enough memory!/
SETERR: .ASCIZ  /?FF-F-.SETTOP error!/
NOLKO:  .ASCIZ  /?FF-F-LOOKUP-Failure/
IOERR:  .ASCIZ  /?FF-W-IO error!/
        .EVEN
```

```
OK:
        CALL DATTIM
        .WRITW  #AREA,#0,#BUFFER,#<BUFEND-BUFFER>/2,#0
        BCS     HRDERR

READY:  .CLOSE  #0
        TST     STAT+4          ;Handler was loaded before?
        BNE     EX              ;If so, leave it!
        .RELEAS #FILO
EX:     .EXIT

HRDERR: .PRINT  #IOERR
        BR      READY
;                           END OF MAIN
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++;

; Ascii header line follows (attention length must be EVEN!!)
; -------------------------------------------------------------
BUFFER: .BYTE   14,15   ;FF, CR
        .ASCII  / *** NKG - AZG *** Printed at / ;YOU MAY CHANGE THIS TEXT
TIMSTR: .BLKB   8.      ;TIME
        .ASCII  / on /
DATSTR: .BLKB   17.     ;DATE
        .ASCII  / *** NKG - AZG ***/ ;YOU MAY CHANGE THIS TEXT
        .BYTE   12,12,15;LF, LF, CR
BUFEND: .WORD   0

AREA:   .BLKW   5


CNVASC: CLR     R0
        DIV     #10.,R0
        ADD     #'0,R0
        ADD     #'0,R1
        CMPB    R0,#'0
        BNE     1$
        MOV     #' ,R0
1$:     MOVB    R0,(R4)+
        MOVB    R1,(R4)+
        RETURN

           .SBTTL  DATTIM
  DATTIM::
           MOV     #TIMARG,R5
           CALL    TIME
  ;
  ; R2=MONTH(1-12), R1=DAY(1-31), R3=YEAR(MODULUS 100)
  ;
  DATE::   .DATE
           MOV     R0,R3            ;See Prog. Ref. 81-V4, pp. 2-26
           BEQ     1$
           BIC     #^C37,R3
           ADD     #72.,R3
           MOV     R0,R1
           ASL     R1
           ASL     R1
           ASL     R1
           SWAB    R1
           BIC     #^C37,R1
           MOV     R0,R2
           SWAB    R2
           ASR     R2
```

```
          ASR      R2
          BIC      #^C37,R2
1$:       MOV      #DATSTR,R4          ;R4=date/tim pointer
          CALL     CNVASC
          MOVB     #' ,(R4)+
          DEC      R2
          MOV      R2,R1               ;*12.
          ASL      R2
          ASL      R2
          ASL      R2
          ASL      R1
          ASL      R1
          ADD      R1,R2
          MOV      #MONTH,R1
          ADD      R2,R1
          MOV      #12.,R2
2$:       MOVB     (R1)+,(R4)+
          SOB      R2,2$
          MOV      R3,R1
          CALL     CNVASC
          RETURN

MONTH:    .ASCII   /January   19/
          .ASCII   /February  19/
          .ASCII   / March    19/
          .ASCII   / April    19/
          .ASCII   /  May     19/
          .ASCII   /  June    19/
          .ASCII   /  July    19/
          .ASCII   / August   19/
          .ASCII   /September 19/
          .ASCII   / October  19/
          .ASCII   /November  19/
          .ASCII   /December  19/
          .EVEN

TIMARG:   .WORD    0
          .WORD    TIMSTR

          .END     START

      SUBROUTINE INCLUD(FILNAM,FARR,IARR,BARR,NINP)

C Purpose: Reading data from a ASCII file, the PARAMETER data file.

C         Supported data types are indicated by key words, these are:
C         REAL, INTEGER, BYTE, LOGICAL and OCTAL.
C         The key word END terminates reading from file.

C  Input: FILNAM; Name of parameter data file:
C                 Byte string 'dev:filnam.typ', terminated with a null.

C  Output:NINP(1); Less than 0, error occurred, else:

C        NINP(1); number REALS red from input file,
C        NINP(2); number INTEGERS and/or OCTALS red from input file,
C        NINP(3); number BYTES and/or LOGICALS red from input file,

C  File layout may be:
C  ------------------
```

```
C  1. Lines with comments containing NO UPPER CASE KEY WORDS and empty lines.
C  2. ONE line with ONE VALID KEY WORD, with comments and immediately followed
C     by:
C  3. LINES OF DATA indicated by previous key word.
C  4. Data file should end with END keyword!

C  Valid keywords:
C       INTEGER -> Integer data follow
C       REAL    -> Real data follow
C       BYTE    -> Byte data follow
C       LOGICAL -> Byte data T(true) or F(false) follow
C       OCTAL   -> Octal data follow (these are stored in Integer array)

C  Note: Max. nr. of values on one line is 10.
C        Max. nr. of bytes  on one line is 80.

C  External routines: FFMT  .FOR; Free ForMat Terminal input.
C                     ASCOCT.MAC; ASCii to OCTal value conversion.
C                     Some syslib routines.

        BYTE LINE(81),GETER,FILNAM(1),BARR(1)
        DIMENSION FHLP(10),FARR(1),IARR(1),NINP(3)

        INP=10
        OPEN (UNIT=INP,NAME=FILNAM,TYPE='OLD'
        1,READONLY,ACCESS='SEQUENTIAL',ERR=2020)

        NINP(1)=0
        NINP(2)=0
        NINP(3)=0

        KEY=0    !Data type unknown jet.

C--------------------------Look for key word--------------------------C

10      CALL GETSTR(INP,LINE,80,GETER) !Read one line from file.
        IF (GETER) GOTO 2000

        IF (INDEX(LINE,'REAL'    ).NE.0) GOTO 100
        IF (INDEX(LINE,'INTEGER').NE.0) GOTO 200
        IF (INDEX(LINE,'BYTE'    ).NE.0) GOTO 300
        IF (INDEX(LINE,'LOGICAL').NE.0) GOTO 400
        IF (INDEX(LINE,'OCTAL'   ).NE.0) GOTO 500
        IF (INDEX(LINE,'END'     ).NE.0) GOTO 3000

        GOTO (10,110,210,310,410,510) KEY+1 !Continue with last key word.

C-----------------------------Get REAL data----------------------------C

100     KEY=1
        CALL GETSTR(INP,LINE,80,GETER)
        IF (GETER) GOTO 2000

110     CALL FFMT(LINE,FARR(NINP(1)+1),N,GETER)
        IF (GETER) GOTO 2010
        NINP(1)=NINP(1)+N
        GOTO 10
```

```
C----------------------------Get INTEGER data-------------------------C

200     KEY=2
        CALL GETSTR(INP,LINE,80,GETER)
        IF (GETER) GOTO 2000

210     CALL FFMT(LINE,FHLP,N,GETER)
        IF (GETER) GOTO 2010

        DO 220 I=1,N
220     IARR(NINP(2)+I)=FHLP(I)
        NINP(2)=NINP(2)+N
        GOTO 10

C----------------------------Get BYTE data----------------------------C

300     KEY=3
        CALL GETSTR(INP,LINE,80,GETER)
        IF (GETER) GOTO 2000

310     N=LEN(LINE)      !Do not copy null byte.
        DO 320 I=1,N
320     BARR(NINP(3)+I)=LINE(I)
        NINP(3)=NINP(3)+N
        GOTO 10

C-------------------------Get LOGICAL(byte) data----------------------C

400     KEY=4
        CALL GETSTR(INP,LINE,80,GETER)
        IF (GETER) GOTO 2000

410     N=LEN(LINE)      !Do not copy null byte.
        DO 420 I=1,N
        IF (LINE(I).EQ.'T') GOTO 422
        IF (LINE(I).EQ.'F') GOTO 424
        GOTO 420
422     BARR(NINP(3)+1)=.TRUE.
        GOTO 425
424     BARR(NINP(3)+1)=.FALSE.
425     NINP(3)=NINP(3)+1
420     CONTINUE
        GOTO 10

C----------------------------Get OCTAL data---------------------------C

500     KEY=5
        CALL GETSTR(INP,LINE,80,GETER)
        IF (GETER) GOTO 2000
510     I=0
520     I=I+1
530     IF (LINE(I).EQ."0) GOTO 10       !"0=0 Line terminator
        IF (LINE(I).LT."60 .OR. LINE(I).GT."71) GOTO 520

        CALL ASCOCT(LINE(I),IARR(NINP(2)+1)) !Convert ASCII to octal value.
        NINP(2)=NINP(2)+1

540     I=I+1 !Skip converted digit string.
        IF (LINE(I).GE."60 .AND. LINE(I).LE."71) GOTO 540
        GOTO 530
C-----------------------------Error occurred---------------------------C
```

```
2000    NINP(1)=GETER !GETSTR error, see syslib routines.
C
C  -1= End of file, -2=Hard error, -3=Too many input characters
C
        IF (NINP(1).EQ.-2) GOTO 3010 !Hard error, skip close!
        GOTO 3000
2010    NINP(1)=-10      !FFMT error.
C----------------------------------Ready------------------------------C

3000    CLOSE (UNIT=INP)
3010    RETURN
2020    NINP(1)=-11      !OPEN error.
        RETURN
        END
```

```
Parameter file for program ERPLOT
LOGICAL  XDIREC (curve expansion V or H), MAXMIN (extrem marking)
F T
BYTE     Title:
This is an example
INTEGER  NRDIG (nr. digits V-scale), NSIG (nr. curves V, -1:NSIG=NCHAN)
1 4
INTEGER  NFILES (nr. curves from different datafiles in one X-Y frame)
1
REAL     AXLEN(1) X-as in cm., AXLEN(2) Y-as in cm.
10 10
REAL     TICK(1) (cm./tick H), TICK(2) (nr. ticks V, negativ = scale factor)
2 -2
REAL     Specifie TBEGIN, TEND, X-TICK(ms.) for VER, if TB=TE default setting!
-500 500 200
REAL     Specifie TBEGIN, TEND, X-TICK(ms.) for SER
-20 80 20
REAL     Specifie TBEGIN, TEND, X-TICK(ms.) for BER
-10 10 4
REAL     Specifie TBEGIN, TEND, X-TICK(ms.) for GER
0 0 0
END
```

DEC INPUT

V5.1 of RT-11 and Its Software Now Run On Professional 300s

RT-11, Digital's most popular operating system, is now available on the Professional 300 Series of personal workstations. With the announcement of RT-11 V5.1, the Professional 300 becomes the lowest-cost system that can run the popular RT-11 operating system and its huge array of software.

Small, fast, and effecient, RT-11 offers both runtime and development environments within a single system. It can handle both realtime and data processing applications, supports a variety of compilers(including FORTRAN-77, APL, and C), and offers a full range of system utilities to make interactive program development easier.

Digital has sold over 80,000 RT-11 licenses. OEMs have sold thousands more RT-11 based systems to their customers. RT-11 users benefit both from Digital's steady development of the operating system and from the huge array of software that third parties now offer.

With Version 5.1, RT-11 users can take advantage of the Professional 300 hardware, including its video bit map capability for precise graphics. Version 5.1 also includes a communications option (VTCOM) that allows any RT-11 system to tie into a host system (either Digital or non-Digital) as an intelligent terminal.

"A Darn Good Idea"

Combining RT-11 and the Professional "is a darn good idea," according to John Crowell, a V5.1 field test user. Crowell, president of Crow4ell Ltd., of Los Alamos, N.M., uses his professional running RT-11 "as an inexpensive dedicated system" for developing application software.

The RT-11/Professional combination "has the mass storage, the communications port, and the transportable media" that he needs, Crowell says.

Crowell also expects to make good use of VTCOM, and TRANSF, (serial line communication program) "partly in transmitting software I've developed to other target systems." He also suspects that "the Pro's bit-map graphics will be a significant feature. I'm anticipating putting it to great use."

Another user, Bob Walraven of the University of California, has done software development under RT-11 by working on a Professional 350 in his home. "It really is good old RT on the Pro," he says.

"The Pro is very quiet, it sits under my desk, and has an excellent keyboard. It's a very convenient system for doing major conversion or development work at home," Walraven says. "I've used the TRANSF program quite a lot. It's very handy".

Multiuser Development on RSX or VAX Systems

Another new product, RTEM-11 V2.0, allows multiple users to develop RT-11 V5.1 software on an RSX-11 or VAX/VMS system. Application programs can be created, edited, assembled, linked, and debugged under RTEM-11 and the executed under RT-11. RTEM-11 creates an RT-11 enviroment on the host system. Users logging onto a host system can be put directly into RTEM-11 through startup files. This allows developers to work with RTEM-11 without having to know the host operating system itself.

Transferring Application Programs from PDP-11s

Customers with PDP-11 systems may want to transfer existing files to MICRO/PDP-11 and Professional 300 systems that use RX50 floppy disks. RT-11 V5.1 provides a simple, inexpensive solution to this problem. The VTCOM program provides ASCII file transfer with a Digital or non-Digital host. The TRANSF program provides ASCII and binary file transfers between RT-11, RTEM-11 (running under RSX or VMS), CTS-300, TSX-PLUS , and RT-11 systems.

New Features

RT-11 V5.1 contains the full set of RT-11 features, which include automated customer installation, Concise Command Language (CCL), User-Definable Command Linkage (UCL), indirect control file processor (IND), single-line editing, logical disk subsetting, virtual memory handler, and the BACKUP utility program.

In addition to VTCOM, Version 5.1 also offers the Transparent Spooling Package (SPOOL), provides simultaneous output to printers (or any other RT-11 serial device) concurrent with other system actions. The user can operate SPOOL without having to command and control spooler actions directly. System operation remains consistent in both the spooled and non-spooled environment.

Another new program, SETUP, uses simple, English commands to set operator modes for the Professional terminal and time-of-day clock, the VT100 family of terminals, and the LA50 and LA100 serial printers. SETUP is especially useful for setting video characteristics by including setup commands in startup indirect command files.

SETUP provides all of the hardware setup features normally used with VT100 terminals.

Professional 300 or MICRO/PDP-11

Both the MICRO/PDP-11 and the Professional 300 Series now run RT-11. Each offers its own advantages. The Professional offers the lowest-cost hardware for RT-11 users, plus its award-winning ergonomic design, and its bit-map graphics capability. The MICRO/PDP-11 offers compatibility with Digital's Q-Bus peripherals, a more standard PDP-11 architecture, memory expansion up to four Mbytes and a multiuser environment under CTS-300 or TSX-PLUS.

Availability

RT-11 Version 5.1 is available now.

*
TSX-PLUS is a trademark of S&H Computer Systems.

USER REQUESTS

I read in some publication a reference to a program, in the public domain, which reads CP/M formated diskettes under RT11. I have no other clue, can anyone help me with this.

I would also like to communicate with CPU's using MODEM77/XMODEM programs. It is no problem using LSTERM (DECUS 11-543) to talk as a terminal to MODEM77 but most CP/M Bulletin Boards ask you to use XMODEM to receive a long file.

[illegible]

Julian H. Unger
AMF Geo Space Corp.
P. O. Box 36374
Houston, TX 77036
(713) 666-1611

DECUS LIBRARY

The following is a list of new and updated programs in the DECUS library that are of interest to RT-11 users.

PARSE/RT: A Flexible Filespec Parser for FORTRAN

Version: V2.0, August 1983

Author: R.W. Bernard, Sandia National Labs, Albuquerque, NM

Operating System: RT-11 V4.0, V5.0

Source Language: FORTRAN IV

Memory Required: Approximately 800 words.

PARSE is a FORTRAN subroutine which makes specification of files used in a program easy, and requires a minimum of operator input. When used with a FORTRAN applications program, PARSE minimizes the amount of typing necessary to specify files by providing default values whenever they are not supplied at run-time. Both default device specifications and extensions may be handled. Once a "root" filespec has been created by PARSE, further filespecs can be created without any user action by means of successive calls to PARSE.

The program TSTPAR demonstrates the use of PARSE, and also provides extensive documentation on the proper use of the subroutine. The subroutine is used with RT-11 FORTRAN IV codes, and calls several RT-11 SYSLIB routines.

An article on the use of the code may be found in the magazine SEXTANT, in the Summer, 1983, edition. This version of PARSE has more functionality than the one included on the DECUS Symposium tape No. 11-SP-53; previous versions are upward with compatible with the new one.

Documentation on magnetic media.

Media (Service Charge Code): Listing (BA), Floppy Diskette (KA), 600' Magtape (MA)

SORT: Fast Memory/Disc Sort/Merge for RT-11

Version: V2A, July 1983

Author: Chester Wilson, Canberra, Charleville, Australia

Operating System: RT-11 V3.0 or later

Source Language: MACRO-11

SORT is a general purpose high speed RT-11 memory/disc sort/merge utility program, capable of coping with files as large as RT-11 can manage. Sorting may be ASCII or alphanumeric, and considerable trouble was taken by the original author to enhance the speed of the sorting.

SORT was written by Darrell Whimp during his student days at St. Peter's Lutheran College, Brisbane. It has been given to DECUS with the kind permission of himself and the computing department at St. Peter's.

Documentation on magnetic media.

Media (Service Charge Code):	Floppy Diskette (KA), 6ØØ' Magtape (MA)

Complete File Sort Utility

Version: V3.Ø, August 1983

Author: Bob Schilmoeller and Paul Styrvoky, St. John's University, Collegville, MN

Revising Author: John M. Crowell, Crow4ell, Ltd., Los Alamos, NM

Operating System: RT-11 V5.Ø

Source Language: MACRO-11

Memory Required: 16KW

Other Software Required: EIS

RTSORT is a substantial revision to Complete File Sort Utility, by Bob Schilmoeller and Paul Styrvoky of St. John's University, Collegville, MN. The program performs a multiple key sort of a data file in either alphabetical or ASCII order. The sort is accomplished via a Tag Array built with the specified sort fields and block and record addresses. A Shell Sort puts the Tag Array in ascending or descending order. The sorted data are written to a file, and, optionally, printed on the terminal.

A maximum of 16 sort fields is allowed. Maximum record length is 2Ø46 bytes. Records must be separated by a /CR/LF/.
In the preparation of this version, no changes in the sorting procedure were made. Revisions consist of the following:
1. Replacement of redundant code with subroutines.
2. Streamlined conversion between binary and ASCII decimal.
3. More efficient file I/O and data storage.
4. Run-time allocation of buffer space and Tag Array storage.

The results of these revisions are:
1. Up to 3Ø% increase in maximum number of sorted records.
2. Size reduction of SAV image from 6Ø blocks to 6 blocks.

A maximum of 16 sort fields is allowed. Maximum record length is 2Ø46 bytes. Records must be separated by a /CR/LF.

Changes and Improvements: Increased record capacity, increased speed, SAV image reduction.

Documentation on magnetic media.

Media (Service Charge Code):	Floppy Diskette (KA), 6ØØ' Magtape (MA)

RETCON

Version: July 1983

Author: Kirk R. Stauffer, Copperweld Robotics,
Madison Heights, MI

Operating System: RT-11 V4.0

Source Language: FORTRAN IV

Memory Required: 10KB

This program displays the contents of the Reticon RSB6320 interface board when connected to a LC series line scan camera. The pixel numbers where the transisions have occured (address) or the number of pixels between transisions (count) can be displayed on a continuous or periodic basis. The user is also able to specify the number of scans that are averaged together prior to display.

Documentation on magnetic media.

Media (Service Charge Code): Listing (BA), Floppy Diskette (KA),
600' Magtape (MA)

FTALK

Version: V1.0, July 1983

Author: Timothy W. Coressel, Rockwell International, Golden, CO

Operating System: RT-11 V5.0

Source Language: MACRO-11

Memory Required: 3KW

Other Software Required: SBC - 11/21 (Falcon) must have the KXT 11-Adoption (Macro-ODT)

Special Hardware Required: Two DL Serial Ports

FTALK is a software package for linking a software development PDP-11 computer to a 11/21 (Falcon) computer used in dedicated type applications. This program allows a user to download stand-alone programs from any mass storage device existing on a PDP-11 computer to the Falcon. It also allows one terminal to communicate to both the PDP-11 computer and the Falcon.

Documentation on magnetic media.

Media (Service Charge Code): Write-Up and Listing (DA), Floppy
Diskette (KA), 600' Magtape (MA)

MACRO-11 Input/Output MACRO Subroutines Library

Version: V1.Ø, July 1983

Author: Rodney Schaerer, St. Mary Medical Center, Long Beach, CA

Operating System: RT-11 V5.Ø

Source Language: MACRO-11

Memory Required: 6144KW

Other Software Required: RT-11 Librarian Utility

This package creates an RT-11 Macro-11 macro subroutine library which contains often used input/output subroutines. Some of the subroutines will clear the terminal's input ring buffer, convert decimal ASCII text strings to binary integers and vice versa, convert a binary integer to an octal or decimal ASCII text string and print the string to the terminal. It will also convert an ASCII format to a RAD5Ø format string, and effect an efficient save and restore regisiters Ø - 5 routine. The subroutines program source files should be examined for the details and operation of each subroutine.

Associated Documentation: RT-11 Macro-11 Language Reference Manual, chapter 7, and the System Utilities Manual, chapters 1Ø and 12.

Documentation on magnetic media.

Media (Service Charge Code): Write-Up and Listing (DA), Floppy Diskette (KA), 6ØØ' Magtape (MA)

TTLIB: VT1ØØ Library Routines

Version: V2.Ø, July 1983

Author: Chester Wilson, Charleville, Australia

Operating System: RT-11 V3.Ø or later

Source Language: MACRO-11

TTLIB is a Library of programs to conveniently control a VT1ØØ terminal in ANSI mode. Routines allow drawing boxes and lines, cursor positioning, screen appearance, video attributes, screen and line clearing, screen and keyboard behavior, graphic facilities, assorted heights and widths, tab setting and clearing, and reporting cursor position.

This package contains the sources for and a constructed library of TTLIB, along with the manual TTLIB.DOC.

The Version 2 release includes TTXON and TTXOFF, and has interfaces for programs written in C for the DECUS C compiler (DECUS No. 11-SP-18).

On reading the manual, you will discover that TTLIB may optionally be set up to be combind with CVLLIB, a general-purpose library by the same author. The .COM files and TTLIB.OBJ files in this package have been set up for a free-standing library - ie. the routines WDT and CSVR are included.

Changes and Improvements: Extra routines; bug fixes; addition of interface routines for "C" programs.

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskette (KA), 600' Magtape (MA)

Extension Routines for MU-BASIC

Version: July 1983

Author: Harald Wiessmann, Wiessmann, Schaltenwurte, Ing. Buro, Reutlingen, Germany

Operating System: RT-11 V4.0 or later

Source Language: MACRO-11

Memory Required: 27KW

Other Software Required: MU-BASIC V2.0 or later

The extension routines enable additional functions in MU-BASIC such as: set time and date, signal wait, input/output of any installed DL line with device time out capability, pack and unpack float values (single precision) to octal and vice versa, clear ring buffer. Except for multiuser I/O functions the extension routines can also be applied under BASIC-11.

Restrictions: The above functions add about 5 blocks to the Basic interpreter. All comments in source are made in German.

Documentation on magnetic media.

Media (Service Charge Code): Listing (German) (BA), Floppy Diskette (KA), 600' Magtape (MA)

RECOVR: An RT-11 Directory Restore Utility

Version: V1.0, July 1983

Author: Paul Gerardi, Schlumberger-Doll Research Center, Ridgefield, CT 06877

Operating System: RT-11 V3B or 4.0

Source Language: FORTRAN IV

Memory Required: 12KW

RECOVR is a routine which modifies the directory blocks of an RT-11 disk that has been INITIALIZED, effectively restoring that disk's files. Unlike the INIT/RESTORE command of the RT-11 monitor, this routine will work on disks which were written under versions of RT-11 prior to V4.0 and on disks written under RT 'compatible' operating systems such as Tekronix's SPS BASIC. All RT-11 block replaceable media are supported by this routine including DECtape-II and PDT devices.

Restrictions: Checked under RT-11 V3B and V4.0. Not guaranteed to work under earlier versions. Magtapes and cassettes are not supported.

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskette (KA),
600' Magtape (MA)

DTX: A Utility for Transferring Text Files and Absolute Binary Images

Version: V1.0, February 1983

Submitted By: Digital Equipment Corporation

Operating System: RT-11, RT-11 Emulators under RSX-11, VAX/VMS, or RSTS/E

Source Language: MACRO-11

DTX is a utility program for transferring text files and absolute binary images to XXDP+ formatted media under the RT-11 operating system or RT-11 emulators under RSX, VMS or RSTS. It supports RX01, RX02, RL01, RL02, Magtape, RK05, RK06 and RK07. It cannot read files from an XXDP+ medium. It is intended for use by persons wishing to develop a program under VMS, RSX, RSTS or RT which will operate under the XXDP+ system.

Associated Documentation: Contact your Digital Sales Representative for the XXDP+ User Manual (AC-F348E-MC). This documentation is not available through DECUS.

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskette (KA),
600' Magtape (MA)

Log: A Log-in, Log-out Utility for RT-11

Version: V1.0, June 1983

Author: Robert A. Malseed, Albuquerque, NM

Operating System: RT-11 V4.0

Source Language: FORTRAN IV

Memory Required: 8600W

This program allows the operator to log computer usage time. The record is kept by log-in category for as many as 15 categories. This information is helpful in determining when to perform power-on-time dependent preventitive maintenance, and in managing operator and computer time.

The operator should run this program to log-in and out as he works from job to job. The program will inform the operator if he has forgotten to log-in or log-out the next time he attempts to do so, and will prompt for a retroactive log-in/out.

The program can display or print monthly and annual summaries of time spent and can reset the monthly and annual time count.

Documentation on magnetic media.

Media (Service Charge Code): Manual (EA), Floppy Diskette (KA), 600' Magtape (MA)

revision
11-SP-18

C Language System, Second Master Release

Version: November 1983

Author: David Conroy, Robert Denny, Charles Forsyth, Clifford Geshke and Martin Minow

Submitted By: Martin Minow

Operating System: RSTS/E V7.2/V8.0, RSX-11M V4.0, RSX-11M-PLUS, RT-11 V4.0, VAX/VMS V3.2, TSX-PLUS V2.2/3.0.

Source Language: C, MACRO-11

Memory Required: 28K Word

Special Hardware Required: DECUS C does not support the PDP-11/40 and LSI-11 "FIS" floating-point unit. Floating point operation requires FPU hardware.

'C' is a general purpose programming language well suited for professional usage. The DECUS 'C' distribution contains a complete 'C' programming system including:
- A compiler for the 'C' language. The entire language is supported except for an emulated (software) floating point, macros with arguments, bit fields, and enumerations.
- A common runtime library ('standard I/O library') for 'C' programs running under the RSX-11 or RT-11 operating systems. By using this library, 'C' programs may be developed on one operating system for eventual use on another.
- A RSTS/E extensions library allowing access to all RSTS/E executive services.
- An RSX-11/M extensions library allowing access to all RSX-11/M executive services.
- More than 20 'C' programs, including a cross-reference lister for 'C' programs, a lexical analyser program generator, cross-assemblers for several microcomputers, and several games.
- Extensive documentation for the compiler and runtime libraries.

All software is distributed in source format. 'C' may be built to run under RSTS/E V7.2/V8.0, RSX-11M V4.0, RT-11 V4.0, VMS V3.1 - 3.2 (compatibility mode) or TSX-PLUS V2.2/V3.0. It may be modified to run on earlier versions of these operating systems and should run on subsequent versions without extensive modification.

Note: There are a few modules which are release-specific and may require modification for earlier or later releases.

(Primarily on VMS compatibility mode). All documentation is included on the magtape but can be ordered in hard-copy. The ordering information is listed below.

Changes and Improvments: Full support for EIS, FPU, improved code generation and improved utility programs.

- Order 11-SP-18 (PC), for the 2400' Magtape, DOS-11 format.
- Order 11-SP-18B (EB), for the DECUS C Language Changes and Compiler Reference Manual.
- Order 11-SP-18C (EC), for the Utility Library Reference Manual.
- Order 11-SP-18D (EC), for the Tool Library Reference Manual.
- Order 11-SP-18E (EA), for the AS Assembler Manual for the PDP-11.
- Order 11-SP-18F (ED), for the Compiler and Library Software Support Manual.

Restrictions: DECUS C supports a subset of the current version of C. Minor problems may be encountered in converting from other dialects of C.

Documentation on magnetic media.

Media (Service Charge Code): See ordering information listed above.

Format: DOS-11

Keywords: Programming Languages
Operating System Index: RSTS, RSX-11/IAS, RT-11, VAX/VMS

new
11-663

MACRO Package for MACRO-11 to Assemble Motorola 680X Code

Version: September 1983

Author: Alan R. Baldwin, Kent State University, Kent, OH

Operating System: RT-11 V4.0

Source Language: MACRO-11

Memory Required: 28KW

This package contains four(4) sets of MACRO-11 macos to interpret Motorola 6800(6802 and 6808), 6801(6803), 6805, and 6809 mnemonics. As is normal in MACRO-11, the output may contain two files; one file contains the binary image of the object in .LDA format, the other contains the assembly listing of the assembled program. A program called LISTER reads .LDA format files and prints the binary data in Octal, Decimal, or Hex formats. A Second program, HEXDCM, can be used to process the .LST file to produce Decimal or Hex listings from the MACRO-11 generated Octal listings.

Several 68ØØ and 68Ø9 programs are included as coding examples, these include: (1) LOADER, a 'DEC' .LDA format absolute binary loader for the 68ØØ, (2) 6821A, a general purpose Motorola 6821 PIA I/O handler (68ØØ code), (3) 825ØA, a general purpose NATIONAL INS825Ø ACE serial-port I/O handler (68ØØ code), (4) MC6845, a general purpose MOTOROLA 6845 CRT controller handler (68Ø9 code), and (5) NS825Ø, a general purpose NATIONAL INS825Ø ACE serial-port handler (68Ø9 code).

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskette (KA),
6ØØ' Magtape (MA)

Format: RT-11

Keywords: MACRO-11, Motorola 68ØX Code
Operating System Index: RT-11

new
11-664

ADCON: A/D Conversions Package for Use with ADV11-C and KWV11-C

Version: V1.Ø, August 1983

Author: G. C. Scott, Trenton, NJ

Operating System: RT-11SJ V4.Ø

Source Language: FORTRAN IV, MACRO-11

Memory Required: 21KW

Other Software Required: DUMP Utility Program

Special Hardware Required: ADV11-C analog-to-digital conversion pc board, KWV11-C programmable realtime clock pc board.

ADCON is a software package which can be used with Digital Equipment Corporation's ADV11-C analog-to-digital conversion and KWV11-C programmable realtime clock pc boards. The characteristics of the software include: (1) digitizations from the 8 differential analog input channels, (2) up to 1kHz sampling rates, (3) software programmable gain, (4) 12 bit data resolution, and (5) dc offset corrections and data calibrations. The package includes the following: (1) READ.ME: user instructions, (2) ADCON.FOR: a FORTRAN IV program which sets up for the A/D conversions, (3) CON.MAC: a MACRO-11 program which controls the A/D conversions, (4) CHANGE.FOR: a FORTRAN IV program which makes available the digitized data in decimal integer format and performs corrections and calibrations on the data, and (5) listing files for the above FORTRAN IV and MACRO-11 programs. Suggestions for software and hardware setup are included for users who require more customized A/D conversion schemes.

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskette (KA),
6ØØ' Magtape (MA)

Format: RT-11

Keywords: Analog to Digital Conversions
Operating System Index: RT-11

new
11-665

PB: Device Handler for Data I/O System 19 Prom Programmer

Version: July 1983

Author: Dipl.-Ing. Michael Iloff, Moses Electronics, Stuttgart, Germany

Operating System: RT-11 V4.0, V5.0

Source Language: MACRO-11

Memory Required: 582 words

Special Hardware Required: Data I/O System 19 Universal Programmer 990-1900

This handler was derived from Digital Equipment Corporation's PC11 high speed paper tape reader in order to allow for device-independent execution of file and command transfer via PIP.SAV to and from the DATA I/O SYSTEM 19 UNIVERSAL PROGRAMMER 990-1900 via a DLV11-J line at address 176520 and vector 320. It needs a running line time clock under a system generated monitor with device timeout feature for reading from the programmer device.

German and English user's instructions are included as PB.GER and PB.ENG.

Note: The system is generated with a device-timeout feature.

Restrictions: Running line time clock.

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskette (KA),
600' Magtape (MA)

Format: RT-11

Keywords: Prom Programmer, RT-11 Device Driver
Operating System Index: RT-11

new
11-631

APL-11 V1.Ø for RT-11, Plus Other Software

Version: October 1983

Author: Doug Bohrer, et.al., Bohrer & Company, Wilmette, IL

Operating System: RT-11 V4, TSX-PLUS 2.2

Source Language: APL, C, FORTRAN IV, MACRO-11

Memory Required: 56KB

Special Hardware Required: FIS or FPP are recommended for APL.

This is a collection of several unrelated programs. The following is a brief description of the programs to be found on the tape. THE FLOPPY DISKETTES INCLUDE ITEMS THREE THROUGH SIX ONLY.

1. Very fast tape backup and restore system. Backup tape is blocked at 1Økb per block and has its own directory. Files can be selectively backed up or restored. Tape writes are double buffered. Written in DECUS `C'. SAV files are included in the distribution.

2. Programs to read IBM and other foreign tapes using RT-11 V4 SYSLIB in FORTRAN and `C' with SAV files included.

3. APL-11 V1, considered more reliable than APL-11 V2. SAV files only. Sources not available.

4. APL utilities include file handling, fancy character bar graphs, print formatting aids and counting type computation functions. Multiple linear regression can use either workspace variables or files for data.

5. FORTRAN/C file handling filter programs to set up APL files, match records from two input files on a key field. SAV files included.

6. FORTRAN subroutines to handle TSX-PLUS shared files wih random access fixed length records. Buffering and locking/unlocking blocks is automatic. Records can span blocks.

Note: Please note that the Floppy Diskettes (KB) contain a subset (items three through six) for floppy systems only.

Restrictions: Shared file routines use TSXLIB (DECUS No. 11-49Ø) which is not included with this package. The sources for APL V1 are not included.

Associated Documentation: FOR APL-11 documentation order the APL-11 V1 RSTS/E Digital manual: AA-5Ø76A-TC from your Digital Sales Representative.

Complete sources not included. Documentation may or may not be included on the magnetic media.
Media (Service Charge Code): Floppy Diskettes (KB),
600' Magtape (MA)

Format: RT-11

Keywords: APL, Shared Files,
Multiple Linear Regression
Operating System Index: RT-11
January 2, 1984

Symposium Tape from the RT-11 SIG, Fall 1983, Las Vegas

Version: Fall 1983

Author: Various

Submitted By: R.W. Barnard, Sandia National Laboratories,
Albuquerque, NM

Operating System: RT-11 V4 and V5

Source Language: BASIC-PLUS, FORTRAN IV, MACRO-11

Other Software Required: If necessary, it will be specified in the program's documentation.

Special Hardware Required: If necessary, it will be specified in the program's documentation.

The symposium tape from the RT-11 SIG contains eleven packages in the form of subdevices, (the packaging method used for tapes from recent symposia). An annotated directory, TAPE.DIR, is the first file, and the file README.1ST explains how to recover the files from within the subdevices. The tape contains the following submissions (listed more-or-less according to subdevice).

1. A program to allow RSTS/E users to recover files from within the RT-11 subdevice files on this and previous tapes.
2. Annotated Symposium tape directories from Fall, 1981, through Fall, 1983.
3. A preliminary version of a User Command Linkage, to allow user-defined commands under RT version 5.
4. A package of programs to allow the transfer of any file (including binary) over serial lines by first converting the file to hex, and converting it back at the other end.
5. Disk librarian - to allow-on-line cataloging and retrieval of all disk directories.
6. A list of telephone area codes which can be accessed on-line. A list of the programming languages available from the DECUS library and their order numbers. A disk verification utility to determine if a particular disk is loaded on a specified drive.
7. Utilities to search a volume and all its subdirectories: time/date stamp printer utility, utilities for TEKTRONIX development systems. Patches to RT V4 or V5 DIR.SAV to force volume ID as the default; a patch to RT V4 PIP to display the input file size and creation date when logging (i.e., "Files Copied...").
8. A collection of routines for data fitting - equation fitting, digital filters, and first-order differential equation solvers.
9. A structured MACRO preprocessor, for the assembler language

SUPER MAC.
10. A plotting package which will generate 2- and 3-dimensional graphs on TEKTRONIX-compatible terminals, and Bausch & Lomb plotters. Written in FORTRAN-IV.
11. The July 1983, DECUS "C" distribution, appropriate for RT-11.

No guarantees are made as to completeness, useability, or quality of the programs on the tape and the material has not been checked or reviewed.

Restrictions: For DSKLIB, the sources have not been released. For PARSE, the sources were submitted as a seperate DECUS submission (DECUS No. 11-562).

Documentation may or may not be included on the magnetic media.

Media (Service Charge Code): Write-Up (AA), 2400' Magtape (PS)

Format: RT-11

Keywords: RT-11 - Symposium Tape
Operating System Index: RT-11

new
11-672

SP: Serial Port I/O Handler

Version: September 1983

Author: Ray Brownrigg, D.S.I.R., Wellington, New Zealand

Operating System: RT-11 V4.0, TSX-PLUS V3.1

Source Language: MACRO-11

Memory Required: Aproximately 1.0KB - 1.5KB

Special Hardware Required: Up to 8 extra serial interface ports. EIS instructions.

The SP handler provides for full duplex I/O on up to eight serial interface ports at one time. In particular, single character transfers are possible, without the overhead of multiple I/O requests, and without the need for privileged mapping on the user program (which would provide access to the I/O page). Also important is the provision of high-speed input to a TSX-Plus program. Two modes of operation are possible. The output only mode, which uses the DC1/DC3 (X-ON/X-OFF) protocol, is invoked by a keyboard COPY command, a FORTRAN WRITE statement, or a .WRITE programmed request in MACRO. The full duplex mode, for which there are various protocol options, is available only through the .SPFUN programmed request in MACRO. At any one time, up to eight serial ports may be active, performing either output or full duplex I/O, any one job may be communicating with more than one serial port, and a number of different jobs may be communicating with the handler. This multiple job capability is particularly useful in RT-11XM and TSX-Plus applications. A user's manual is included as documentation on the media.

Restrictions: RT-11SJ must have Device I/O Timeout support enabled.

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskette (KA),
600' Magtape (MA)

Format: RT-11

Keywords: RT-11 - Device Handler, Data Communications
Operating System Index: RT-11

Upcoming Symposium Information

UPCOMING US SYMPOSIA DATES

1984	Spring	- June 4-8	Cincinnati, Ohio
1984	Fall	- December 10-14	Anaheim, California
1985	Spring	- May 27-31	New Orleans, Louisiana
1985	Fall	- December 7-11	Disneyland Hotel, California
1986	Spring	- April 27 - May 2	Dallas, Texas
1986	Fall	- October 6-10	San Francisco, California
1987	Spring	- April 27 - May 1	Nashville, Tennessee
1987	Fall	- Nov. 29 - Dec. 4	Anaheim, California

CINCINNATI PREVIEW

Ned W. Rhodes

I would like to preview the sessions that will be presented at the Cincinnati symposia. This is being written in advance of the scheduling meeting, so it cannot be considered final, but I anticipate that the final schedule will be close to the one I am presenting below.

If you have never attended a symposia, let me encourage you to try and do so. They are worth the effort. You will be able to meet the RT-11 development team and exchange information with other RT-11 users. It is a good place to get your problems solved.

If you have a session you would like to see given, write me a letter or give me a call. I will either "volunteer" you to give the session, or I will attempt to find someone who can. The symposia is very much a user driven event.

And, if you will be attending future symposia, and are interested in helping out, let me know as I am always in need of session chairmen for the sessions. The job of the session chairman is to introduce the speaker and to fill out an evaluation form. It is very little work, and you get to meet new people. If you are interested, let me know.

The sessions by day follows, but see the disclaimer above.

MONDAY

The day will start with the RT-11 SIG roadmap and business meeting. Next the RT-11 group will present the RT-11 Product Panel and Overview of the Current Release, RT-11 Language and Layered Product Panel and How RT-11 was Put on the Professional. The Monday night sessions will look at Enhanced Command File Processing for RT-11, a user written Lock Manager for RT-11 and a description of RT-11 Library Files. The evening will conclude with a look at TSX-Plus Shared Run-time Systems. This will be an expanded presentation (i.e. with examples) of the one that was presented in Las Vegas.

TUESDAY

Tuesday's sessions will include a paper on how to develop RT-11 device handlers. Then there will be two FORTRAN oriented papers, one on improving the performance of RT-11 FORTRAN programs, and a repeat of Ron Trellue's paper on Accessing Memory above 56KB from RT-11 FORTRAN. The Digital presentations will include a comparison of RT-11 to CP/M and MS/DOS as well as an IND applications tutorial.

WEDNESDAY

This day will contain a number of diverse session presentations. Jack Peterson will repeat his paper on Combatting Flash Floods with PDP-11s. Dan Kingsbury and Killer will conduct the TSX-Plus Magic session, which will be followed (but not necessarily connected with, nor is any connection implied) by a paper on TXS-Plus internals. For the person interested in the PRO, two sessions on Wednesday will consist of a PRO-350 Question and Answer session that will consist of RT-11 5.1 test site users followed by a Digital tutorial on how to transport RT-11 programs that work on PDP-11's to the PRO-350.

THURSDAY

Thursday is usually a long day and filled with RT-11 sessions. John Crowell will present his paper on the RT-11 User Command Linkage. Two TSX-Plus papers will look at RT-11/TSX-Plus compatability issues and TSX-Plus Real-time I/O techniques. The ever popular RT-11 User Application Workshop will happen some time during the day and will feature short user presentations on interesting RT-11 applications. The evening festivities will start with two user application papers, followed by a KED feedback session. The evening will finally end with the RT-11 Users Presentation session. In this session, all topics are discussed and answers (both right and wrong) are offered. This session is truly a give and take and is entirely user driven. Costumes are not required, but often worn.

FRIDAY

The symposia concludes with the RT-11 User Feedback session that features responses from Digital concerning "wish list" items that were submitted during the week. Planning for the next symposia is then started at the wrap session.

I hope that you can see some sessions that will be of interest to you and I hope that you can make the convention.

The following RT-11 related sessions at the Spring 1984 DECUS Symposium at Cincinnatti will be scribed for the 'Mini-Tasker'.

1. RT-11/TSX-Plus Compatability Issues
2. Accessing Memory Above 56Kb From RT-11 FORTRAN
3. RT-11 On The PRO-350 Panel And Q & A
4. TSX-Plus Internals
5. How To Develop RT-11 Device Handlers
6. RT-11 Product Panel And Overview Of Current Release
7. IND Applications Tutorial
8. Using TSX-Plus Shared Run-Time Systems
9. Improving Performance Of RT-11 FORTRAN Programs
10. TSX-Plus Real Time I/O Techniques
11. TSX-Plus Magic

If possible, these RT-11 related sessions will also be scribed for the 'Mini-Tasker'.

1. A User Command Language With TSX-Plus - Like Command File Processing
2. Combatting Flash Floods With PDP-11's
3. User Written Lock Manager For RT-11
4. A FORTRAN Prompting Library
5. A Challenging Data Acquisition Problem Solved By RT-11
6. RT-11 User Application Workshop
7. RT-11 User Command Linkage
8. RT-11 Users Speakout
9. RT-11 Library Files
10. How RT-11 Was Put On A Professional 350
11. Digital Compares RT-11 To CP/M And MS-DOS
12. RT-11 Language And Layered Product Panel
13. Fast Access Of Extended Memory For Nuclear Magnetic Resonance (NMR)

Past Symposium Information

title   RT-11 DIRECTORY STRUCTURES INTERNALS

speaker Martin Gentry
        Digital Equipment Corporation
        Maynard MA

chair   Jack Crowell
        Los Alamos Labs
        Los Alamos, NM

scribe Gavin Perry

The information from this paper is documented in the Software Support Manual, Chapter 9.

RT-11 has the simplest directory of all the DEC operating systems. All RT-11 files are contiguous. File space is allocated in 512 byte units called blocks. An RT-11 file can not be extended unless there is empty (free) space following the file on the disk. RT-11 files can not span volumes. It is due to these characteristics that the directory structure is so simple.

Every device is divided into blocks. At least the first 8 blocks are reserved (for larger devices up to 60 more blocks may be used for directory segments). The first block is called the boot block and contains the primary bootstrap on a bootable volume. If the volume can't be booted RT-11 will come back with "NO BOOT ON VOLUME" error when you try to boot that device. The second block is the home block and is discussed below. Blocks 2 through 5 are reserved by DEC and contain the secondary bootstrap on bootable volumes otherwise they will be full of garbage. Block 6 is the beginning of the directory and the directory continues two blocks per segment to the end of the directory. The next block will be the first block of the first file on the device. The size of the directory is set when a volume is initialized. The program which initializes the volume (DUP) decides a default size for the directory dependant on the size of the device; this can be overridden with the /SEG:n switch to the INITialize command. (e.g. INIT/SEG:4 will force 4 directory segments (8 blocks) to be put on the disk.)

The home block contains a bad block replacement table of 128 bytes which allows bad blocks on a device to be replaced with spare good blocks. Not all devices support this option; RL02s do but floppies don't. On an RL02 there are 10 blocks which can be used for bad block replacement; larger disks can have more spare blocks. The table entries consist of the bad block address (a 16-bit word) followed by the replacement block address; a 0 word ends the table. The boot, home and directory blocks must be good to use a volume for obvious reasons; any data block can be replaced. (The next entry in the home block (octal 204 to 251) is the INIT/RESTORE data.) When a volume is initialized the first directory entry is copied to this area before it is zeroed out. Thus a mistakenly initialized volume can be restored by simply copying the information here back into the first directory segment. BUP (V5 Backup Utility Program) sets to use bytes 252 through 273. RTEM data is stored in bytes 700 to 703. The pack cluster size is in the word at 722-723 and is always 1 for RT-11. The next word contains the block number of the first segment (which for RT-11 is always block 6). Following this is the system version (directory type) in RAD-50 ("V05") in version 4 and version 3B it was "V3B" this word should not have been changed since the directory structure wasn't changed. But since it was version 4 and earlier can not read a disk initialized under V5. If you have to use volumes on both V5 and V4 systems initialize the volumes on the V4 system and you can use them on either system. If you forgot to do this and have a V5 disk to read on a V4 system (and no access to the V5 for a simple copy) then it is still possible to get at that disk. Using SIPP open the volume and patch the bytes in block 1 offset 726 to read "V3B" in RAD-50. The next bytes contain the volume ID in Ascii. The default volume ID is "RT11A ".This is followed by the owner name which defaults to 12 blanks. The next field is the system identification "DECRT11A " and finally there is the block checksum. Except for the block replacement table most of the home block is unused by most of the normal RT11 utilities.

The directory itself is modified by the user (User Service Routines), DUP (Disk Utility Program), and BUP (Backup Utility Program). JOAT (Jack-of-All-Trades) in RTEM, RSTS File Transfer (FIT) and RSX/VMS File (X)transfer (FLX) can also modify the directory of an RT-11 volume.

The directory starts in block 6 and may have 1 to 31 segments. There are two blocks per segment (1024 bytes). Each active segment has a header and one or more entries. A directory segment has a five word header, up to 72 entries and an end of segment entry. The segment header's first word is the total number of segments (only used on the first segment). The next word is the link to the next segment (the segments are not necessarily consecutive on the disk because a full segment can be split). This word will be 0 if this is the last segment. For the first segment only the next word contains the highest active segment.

The next word tells how many extra bytes are in each directory entry (this is normally 0 to allow the most entries in the directory but is user definable with the INIT command). Finally the last word is the first data block number for this segment.

Each directory entry contains at least 7 words. The first word is the status word which tells whether this file is tentative, empty, permanent, or the end of the segment. The next two words contain the file name in RAD-50 followed by a word with the file type in RAD-50. (RAD-50 is a way to squeeze 3 characters into a word instead of allowing only 2 (bytes) as in ascii; that's why file names can only have letters and numbers in the name since RAD-50 is limited to 50 octal (40 decimal) characters in the character set.) The next word is the file length in blocks. For a tentative file the next word contains a byte for the job which is attached to it (FB or XM) and the other byte contains the channel which is attached to that file. For other than tentative entries this word is unused but still reserved by DEC (I bet if we tried real hard we could get them to put the createion time here if they don't have any other plans for the word in permanent files.) The next word is the creation date in DEC format (for permanent and tentative files only). This is the end of the entry unless there are optional extra words included in the entries as specified by the first segment header's third word.

The main difference between the different entry types is the status word and which of the other words are used or ignored. A permanent entry has the status word 2000 octal or 102000 for a protected file. A tentative entry has 400 as it's status word. An empty entry has the status 1000, a length, and the rest is ignored. When a file is deleted the status word is simply changed from 2000 to 1000. That is why it is so easy to undelete a file in RT if it has been deleted in error. Typing DIR/DEL will display the names of files that have empty entries and their locations and size in decimal. A CREATE/START:nnn filename.typ[size] command will then recreate the file. This must of course be done before the file has been written over with another file or the directory squeezed which cleans out all the embedded "empty" files. Since the consolidation process puts all the contiguous empty files together under one entry, the files shown as deleted may be bigger than the file that was deleted, or the file name may not appear if it had an empty file preceding it in the directory before it was deleted. The moral is be sure to check (with DUMP or TYPE) to be sure that the file recovered is the same version as the file wanted.

Under RTEM there is a filler entry which looks like a protected file (status word 102000) except that the length is 0. This entry marks the end of the shared area of RTEM. The end of segment entry consists of the status word 4000, the rest of the entry is unused.

For file operations RT provides the following programmed requests which are performed in the user. These are the routines that the monitor commands (e.g. CLOSE) and utilities (e.g. PIP) use to manipulate the directory:

.CLOSE	Causes the tentative entry open on the channel speecified to become permanent. If the requested Job-channel doesn't exist it returns an error.
.DELETE	Removes a permanent entry by changing it to an empty (the status word is changed to 1000 octal).
.ENTER	Allocates space for a file and createes a tentative entry.
.FPROT	Changes the file protection bit in a permanent entry.
.LOOKUP	Searches the directory for a permanent entry with the name specified.
.PURGE	disassociates a channel from a tentative entry.
.RENAME	Changes the file name or type words in a permanent entry.
.SFDATE	Changes the createion date in a permanent entry.

The Disk Utility Program DUP has the following file operations:

CREATE	Make file and set size.
EXTEND	Make a file larger if possible.

INITIALIZE	Make directory empty (createe new directory).
SQUEEZE	Compact the directory and the files (consolidate free space).
UNINITIALIZE	Restore old directory (INIT/RESTORE).

title RT-11 XM New User

speaker Greg Adams
Digital Equipment Corporation
Maynard, Mass.

chair William Walker
Sandia Labs
Albuquerque NM

scribe Gavin Perry

RT-11 XM, the extended memory monitor, has been improved. The PC version of RT can run the FB monitor or XM, but not SJ. Because of this limitation DEC has decided to make the XM version as easy to use as possible. Since the PDP-11 is a 16 bit machine with a 16 bit Program Counter (PC) it can only address 64k. Each byte is addressable for 64k bytes or only 32k 16-bit words of address space for a program (k = 1024). The basic PDP-11 bus layout is to have memory from 0 to 157777 octal (56K) and the I/O page from 160000 to 177777 (8K). At the top of memory is the SY: handler and below that is RMON. Below RMON and any other loaded handlers is the free (user available) memory. This address is called SYSLOW.

The problem is that you don't know the value of SYSLOW. (It varies from system to system.) A save file disk image goes from 0 to .LIMIT. Under SJ when running an image, it is mapped directly location for location into memory. (If .LIMIT is greater than SYSLOW the program is too big to load and run.) Normally there is space between the .LIMIT (top of the program) and SYSLOW (the bottom of the system). The programmed request .SETTOP allows the space between the system and the program to be accessed in SJ or FB for variable space, etc. This is done automatically in BASIC or FORTRAN. In MACRO the program will have to figure out how much space is there. The linker provides the constant .LIMIT which is the actual top of the linked program including any library routines that were linked in at the end of the program. After calling .SETTOP the program can subtract .LIMIT from R0 to find the amount of free space.

There are three different sizes of memory available for the various PDP-11 machines. The PDP 11/20 and the T11 are unmapped and thus provide a total of 16-bits or 64K bytes of memory. The PDP 11/23, 11/34, 11/44 provide the extended 18 bit bus which can address 256K bytes of memory with the help of a KT11 Memory Management Unit (MMU). The PDP 11/70, 11/23+, and 11/73 machines have extended the bus to 22 bits for a total addressable space of 4M bytes (4 X 1024 X 1024) and again a KT11 or equivalent MMU. Under XM the program must still fit into 64K bytes of address space (including the I/O page) but addresses of SAVe files are no longer mapped one for one into memory unless it has privileged mapping. The save image goes into virtual space somewhere in memory and there is much more free space. Only the .SAV part of the memory is visible in the mapped region; any attempt to access outside this region gives a MMU fault. There are several ways to access the extra memory in an extended bus system. FORTRAN virtual arrays can reside in extended memory. The VM: handler can be used to allow the extra memory to look like a very fast disk. (The VM: device is bootable under SJ or FB but not XM.) The program can issue programmed requests to access extended memory. (This can cause the system to crash if the monitor gets hit by the data put into upper memory.) Finally, virtual overlays allow the program to act like a normal overlayed program except that instead of swapping the overlay in from the disk to execute it, the memory map is switched to map the segment into the program space. This is a much faster way to get large programs to run. To link a program with virtual overlays, just use the /V:n switch at link time instead of the /O:n switch for the overlay segments. The main difference between normal overlays and the virtual overlay is that since a new copy is not read in at each call to the segment, if the code is self-modifying or contains variables, they will need to be reinitialized, since changes stay changed. The normal restriction that subprograms in the same overlay segment can't call each other still applies.

A virtual overlaid job contains 'VIR' in RAD-50 in word 0 of the .SAV image, and .LIMIT is set to the next 4K word boundary. This can cause some big holes in the program space. One simple solution is to concatenate all the overlays into one big segment which may take less space than several consecutive segments on 4K word boundaries. This way the program will not be swapped out and it can run very fast even though it is bigger than would fit into the normal 56K bytes available.

Using the programmed requests .CRAW, .CRRG, and .MAP, the program can access extended memory itself. With .CRAW an address window into extended memory is created. A .CRRG request asks for a region of a specified size in extended memory. Finally a .MAP request will associate a program virtual address with a physical address. This method of accessing extended memory is for the more experienced MACRO programmer. More information is available in the Software Support Manual.

title Ethernet LAN for UNIX and RT-11

speaker John Barr
University of Montana
Missoula, Montana

scribe Bill Jackson

The University of Montana had implemented an Ethernet connecting a VAX 11/750 and multiple PDP-11's. The network uses 3Com hardware for Unibus and Q-bus systems. All software was developed at the University because of the unique requirements of their applications. The net provides services for both UNIX and RT users. Services include:

virtual Terminals - unix users can access the VAX from any terminal location on the network.

Virtual Disks - RT users can access areas of the VAX's unix file system as logical devices. This also leads to sharing of 'disks' among RT systems.

Print Server - All users can transfer files to the VAX for spooling onto the line printer.

Time and Date Server - The RT users have their system dates and time initialized by the UNIX system when they (RT) boot up.

Graphics Display Server - Users can share access to the graphics device.

The implementation of the network software was done by an undergraduate and took approximately 2 student months (4 man months?). The most significant problem encountered was the inability of processes in the UNIX system to block while waiting for multiple events. To solve this problem a 'rendezvous DRIVER' was constructed which allowed for the central Ethernet process to communicate with working processes through pipelines to transfer the necessary control information.

The only software required on the RT system were the application tools to read from the Ethernet and interface to the users, and the Ethernet Device Driver. The Ethernet device driver is very similar to a standard disk driver, but it does it's I/O with the Ethernet Adapter.

Problems exist in the basic philosophy of the network if higher level functions were to become required. The basic attitude in the current system was to get something to satisfy the immediate needs with available hardware and some quick' software techniques and gain experience with Ethernet. Future development is expected to reinvent existing software while designing in more sophisticated mechanisms.

title HOW THE J-11 STACKS UP AGAINST OTHER PROCESSORS

speaker David E. Bachschmid
Household Data Services
Reston, VA

scribe Gavin Perry

The new DEC PDP-11/70 on a chip, the J-11 was compared with the top of the line microprocessors from other major manufacturers. These include National Semiconductor 16032, Zilog's Z8000, the 8086, 80186 and 80286 from Intel, and Motorola's 68000. The features and architecture of each of these processors were described and then the results of several benchmarks were presented.

The J-11 features resident memory management with 4 megabytes of address space, directly addressing 64k bytes of program space and 64k bytes of data space. There are three levels of memory protection in a multi-user multi-tasking environment: Supervisor mode, Kernal mode and User mode. A comprehensive orthogonal instruction set of 140 instructions includes floating point (FP11) on the chip. It is compatible with the PDP-11 system software since it implements a superset of the 11/70 instruction set.

The high performance system oriented architecture includes a pipelined architecture and a 32-bit internal data path. The J-11 supports optional cache memory (a must for high performance systems), multi-processor operation, and a coprocessor interface. Other features in this high performance package include microdiagnostics, clock generation, 22-bit console Octal Debugging Technique (ODT) and DMA arbitration.

The J-11 is a 60 pin 1.4 inch DIP consisting of two silicon chips on a single eight layer ceramic chip carrier. The first board level product (the 11/73) runs at 3 to 5 times the speed of an LSI 11/23. The full 11/70 instruction set is implemented including 2 register sets and 6 floating point accumulators (64 bits). A CPU error register identifies illegal halts, address errors, non-existent memory, I/O bus timeouts and stack violations.

The National 16000 series is implemented in a three chip set. The CPU chip which is housed in a 48 pin package, has 8 32-bit general purpose registers, 6 special purpose 24-bit registers (PC, Stack Base etc) and two 16-bit registers (status and module). The NS16081 floating point unit (FPU) has a 32-bit floating point status and eight 32-bit registers. The FPU provides **32** and **64 Bit IEEE FPU format** but not the **80 bit format** and comes in a 24 pin package. The NS16082 Memory Management Unit translates the virtual addresses of the CPU to physical addresses in main memory and contains a 32 entry translation cache along with a register file block and a debugging block. A virtual that is in the associative cache can be transported to physical address in 1 clock cycle (100NS), a page table entry fetch takes up to 20 cycles and finally a page swap is time dependant on the disk and DMA's speed. The MMU comes in a 48 pin package. There are plans for full 32-bit processors, NS32032 and NS32132.

The Zilog Z8000 comes in two versions a 40 pin package (Z8002) and a 48 pin package (Z8001). It has 8 16-bit registers each of which can be addressed as separate 8 bit registers (low byte, high byte), 6 16-bit address registers and 5 special purpose 32-bit registers. The Z8000 provides segment, offset and displacement word to calculate effective addresses. The Z8000 is being used in process control and military applications but does not seem to be finding its way into a lot of general purpose computers as its predecessor the Z80 did.

The Intel 80x86 family provides an upward migration path from the 8080 through the 8085 and 8088 CPUs. The 8086 comes in a 40 pin DIP. This small size is provided by multiplexing the address and data lines for the lower 16 address lines which does increase memory access time to some extent. As with the Z8000 the four main 16-bit registers can also be split and used as 8-bit registers. In addition there are four index registers, a program counter a status register and four (address) segment registers. At the source level 8086 is upward compatible with 8080, 8085 but not the Z80 code. The object code of an 8086 will run on the 8088, the 80186 and the 80286. Digital MACRO users should note that 8086 assembly language has reversed source and destination object fields. The lowest member of this chip family the 8088 which has an 8-bit data path is used in a well known PC from another major computer manufacturer so we may be hearing more about this family of processors.

The Motorola MC68000 CPU is the most similar to the DEC CPUs of any of the competition. It is provided in a 64 pin package so that separate address (24) and data (16) lines may be brought out. The 68000 provides for overlapped fetching and decoding of instructions (pipelining). There is on chip bus arbitration logic. To provide 16 mbyte address range, 24 bits of the 32 bit program counter are brought out. There are 17 32-bit registers in the 68000 (the 68010 provides an extra register for apointer to the interrupt vectors). A special line on the 68000 allows the use of all the older 6800 (6500) hardware and peripherals (e.g. 6821 PIA, 6850 ACIA, 6522, 6551, etc.). There are 8 general purpose data registers, 7 address registers, two stack pointers a program counter (only 24 bits go external for 16 Mbytes of addressing) and a 16 bit status register consisting of a user byte and a system byte. As with the VAX data can be byte wide 16-bit words or 32-bit long words. In addition any bit can be set, tested, or cleared. Eight levels of interrupt priority can be used with a simple 8 to 3 priority encoder chip. The 68010 is expected to come in a 16MHz version though currently 8, 10 and 12.5 MHz versions are available. There are to be 4 versions of the 68000: a low cost 8-bit data path version (68008), the standard 16-bit version (68000), a virtual memory 16-bit version (68010), and a very high performance full 32-bit version. All are fully compatible at the object code level. The 68020 is expected to have parts for testing in January or February 1984. There is a memory management unit for the 68000 (the 68451), a fast floating point software subroutine library (M68KFFP) and an 80-bit wide floating point processor (68881) available, as well as many 680XX support chips including DMA controllers. The FPP will support conversions from the IEEE single, double and double extended precision formats. The greatest weakness of the 68000 family seem to be the associative lookup mapping scheme of the MMU (68451) which is quite slow compared to the power of the CPU. Some OEMs have been using the 68000 with their own or other third party MMUs. There are at least 17 new systems available that use the 68000 processor and run UNIX or a UNIX-like operating system.

Several benchmarks were tried on the various processors described above and a comparison was also done based on the spec sheets for various instructions. Referring to the tables that follow it becomes obvious that depending on what benchmark is used the different processors change places as to which is fastest. The 68000, J-11 and 80286 have comparable speeds for many instructions, for certain benchmarks the 68000 is nearly as fast as the VAX-11/780. The important thing to remember is that raw speed isn't everything. Availability of software and the need for speed in the particular application are more important considerations. One of the slowest CPUs available, the Z80 is still be built into hundreds of CP/M based systems simply because of the huge base of (relatively mediocre or even worthless) software that is available. If your current application runs on a PDP11 there is nothing like the J-11 (as the 11/73 or as incorporated into future products) to provide for increased throughput and enhanced functionality.

title TERMINALS RAP SESSION

chair David McDonald
DIGITAL Terminal Products Committee

scribe J.P. Dillenger

And what would you like to see in a terminal? DECUS has another voice. The terminal users grouped together in a symposium session to 'rap' about their needs. Digital's terminal products committee was there to hear it all. The key issues were to cover product future features, terminals with in the company,and marketing and design prospects.

Applications of video text, word processing, inquiry response, data entry/retrieve, and packaged solutions were to be discussed in a open forum type of session. But, as the terminal was never quite separate from the systems before, this session wasn't packed with discussion. Resembling more of a question and answer panel, the responses were limited to a person's request and a show of hands/applause.

The first question, asked by DIGITAL's panel, "What standard features would you like?" was the last question. The discussion started from there and, with a variety of responses, opened a few viable markets for DIGITAL's terminal developers.

The answers included low replacement costs of parts, higher resolution, multiple in-put, multiple out-put, windowing with more intelligence, split screen or two terminal displays.

Graphics and keyboard changes made the hit requests. As in the home market, the majority of persons at the session couldn't accept that Digital was still in the "development" stages of graphics software. Keyboard changes seemed, to this computer novice, a respectable request. To be able to change from one set of reference keys to the next should be included in the software package; the user shouldn't be expected to learn new keyboards everytime there is an advance in the system. It is difficult enough to understand the limits and depths of the new systems.

But, the biggest request was for a bigger screen. Although the crowd was not able to reach a medium figure, the highest need was 2000 X 1500 pixels and the lowest was 1000 X 1500. The amounts that these people were willing to pay also varied. For the most part it was that if DIGITAL provided them, they would purchase them.

One response was that a multiple set-up with host response and inquire statement to update internal state was needed to be able to find out if an internal change had been registered. One response that received applause was for an advanced editor. Also, when the request was made for a anti-glare screen with sharp characters, the crowd roared.

By this time the question had been worn down. But, the responses were still flying. The panel tried to redirect some of the queries by asking people to qualify their answers as to whether or not the needs were standard. Yet the crowd had changed its tune.

Instead of repeating requests to the designers, the next several persons addressed the poor post-purchase support for terminal operations. Applause and nods made the panel begin to take notes. "Product presentations need to focus on results...not plans," said one dismayed user.

Touch sensitive screens were touched on. Dumb terminals made the grade, they could be used as inexpensive access when just simple windows were needed. One user requested that graphics be set on the host instead of making the user depend on different software.

By the end of the session, many ideas had been introduced and discussed. The panel members appeared to be hearing it all, but only time will tell. Priorities were questioned but never really addressed beyond the need for increased screen size.

The terminal users have quite a demand for new products. Their petitions would hold more clout if they would stand together. The panel was looking for a majority voice to inable them to make user oriented plans. Maybe what DECUS needs is a terminal users sig.

speaker	Phil Neray Digital Equipment Corporation
chair	Gary Beaner Rainbow Computing Northridge, California
scrib	Paul E. Triulzi

This session covered the capabilities of the P/OS for graphics. The various hardware were discussed and descriptions of the bitmap hardware was presented. In addition, a discussion of graphics protocols available in P/OS from DEC and other vendors was presented.Those present were treated to slides of a wide variety of pictures and graphs produced on the P/OS system.

The standard system for P/OS graphics is a PRO-300 which consists of a PDP-11/23 PLUS chip set, 512 Kbytes of RAM, and a 10 Mbyte Winchester disk. The software is P/OS, RT-11, or UNIX. The bitmap hardware is integrated in the system. It has a resolution of 1024 X 256 bits which results in 960 X 240 pixels. An extended bitmap option (EBO) is also available which is required for color graphics.

The primary software development tool has been the CORE Graphics Library (CGL). New features announced for the CGL are:

Batch
Changeable cursor shape
Adjustable line thickness
Polygon-fill primitive

A discussion of the General Image Display Instruction Set (GIDIS) was presented. GIDIS is the graphics "fast path" and is useful when speed and compactness are main considerations. It was announced that GIDIS will be enhanced.

The bitmap hardware was discussed in detail. Some of the applications suited to direct bitmap access are:

Games
Real-time data displays
Digitizing photographs
Raster operations
Specialized OEM operations

Disadvantages of directly accessing the bitmap are:

Increased programming effort
The software is more device dependant.

Some of the hardware that are available for use with the PRO-300 systems include:

LA50, LA100
Pen plotters
RGB Color Cameras
Inkjet and thermal printers
Any RS-232 device through the printer port

Some new graphics packages available are:

FINGRAPH - a highly sophisticated, yet easy-to-use visual management system for managers and financial professionals.

MAPS/Pro Graphics - a fully functional business graphics system for the professional with easy to use question and answer dialogue.

ATHENA/graph - created by Athena Systems, Inc, is a package for drawing general business graphics. This is a high-end product with very crisp looking graphics. The package is easy to use because it provides a set of 25 standard displays, which you then edit to get your desired graph. Novice users can be expected to produce relatively sophisticated graphs at the end of their first session. Experienced users can generate high-quality presentation graphics.

DECGRAPH 300 - A German package similar to ATHENA that is currently being translated for use in the U.S.

Futures for the Professional 300 system include the following:

NAPLPS terminal emulator
Tektronix-4014 terminal
Graphics editors
Input support in CGL, PRO/GIDIS
Solution to serial port limit
More DEC graphics hardcopy devices
GKS - possible ANSI standard
IVIS - moving video coupled with CGL graphics

RT-11/TSX-PLUS COMPATIBILITY ISSUES

Jack J. Peterson
Horizon Data Systems
Richmond, Virginia 23233

ABSTRACT

RT-11 is the most compact and responsive of the DEC PDP-11 operating systems, capable of supporting multiple tasks but only a single user. TSX-Plus, a proprietary product of S & H Computer Systems, Inc., extends RT-11 functionality into a multi-user environment. Although TSX-Plus is generally compatible with RT-11 at the .SAV level, there are some differences users and programmers who work with both systems should be aware of. This paper, reflecting the experiences of the author, describes some of these differences.

INTRODUCTION

RT-11 is Digital Equipment Corporation's most compact and responsive operating system. Intended for real-time applications, RT-11 supports up to eight simultaneous tasks but only one user. TSX-Plus, a proprietary product of S & H Computer Systems, Inc., purports to extend RT-11 functionality into a multi-user environment while maintaining compatibility at the memory image (.SAV) level. For the most part, this objective is met; however, there are differences which can affect users and/or programmers who work in both environments.

Incompatibilities are considered from two perspectives: people affected and incompatibility type. Incompatibilities affect users and programmers. Users, who interact with a system, are affected by variations in commands, command formats, options, and results. Programmers, who write applications, utilities, and handlers, are affected by variations in operating system services, especially when their programs must run under both RT-11 and TSX-Plus.

Incompatibilities are further divided into three types: shortcomings, extensions, and gotchas. Shortcomings are RT-11 features not supported by TSX-Plus; if you are attached to such features, you will find transition to TSX-Plus an unhappy experience. Extensions are TSX-Plus features not defined in RT-11; if you use these features indiscriminately, you will be unable to retrofit TSX-Plus applications to RT-11. Gotchas are features present in both systems which do not produce the same results; these require extra care when preparing user manuals, training new users, and writing programs to run in both environments.

Unfortunately, there are many versions of RT-11 and TSX-Plus in active use; the compatibility combinations are prohibitively many. To simplify, RT-11 Version 5 and TSX-Plus Version 4.1 are the standards on which this paper is based. Many of the issues discussed herein are equally applicable to early releases of both.

Many TSX-Plus shortcomings can be overcome or modified by various hacks, which may result in a system not supported by S & H or one incompatible with later releases. Unless you have an excellent background in system software and are capable of supplying your own software support, think twice before installing a dime store solution to a complex problem.

USER-ORIENTED SHORTCOMINGS

Most user-oriented shortcomings of TSX-Plus center on some unsupported tools. For example, TSX-Plus does not support the single line editor (which is being greatly improved in RT-11 Version 5.1), does not permit device formatting (a hardship for diskette users), and does not support error logging by device handlers (very useful on flaky systems). TSX-Plus also fails to support BATCH, a condition that many would call a feature rather than a shortcoming.

Those who maintain several TSX-Plus systems find the lack of a user-friendly SYSGEN procedure a major shortcoming. A TSX-Plus SYSGEN involves editing a MACRO-11 source file to change system parameters and define operational configurations via macros. This method compares unfavorably with the interactive IND-based procedure used by RT-11 Version 5.

TSX-Plus has several limitations regarding handlers. First, it requires all supported handlers to exist at start-up time in both RT-11 and TSX-Plus flavors. If you need to write a TSX-Plus handler, you must also write an installable RT-11 version, at least in skeletal form. All TSX-Plus handlers must be memory resident at all times. Since TSX-Plus and its handlers may not exceed 40K bytes in length, this limitation makes it possible to generate a TSX-Plus system too large to run. There are some workarounds to this problem, most notably generating several TSX-Plus systems, each having a unique handler complement, then running whichever supports the handlers currently needed. This is not always satisfactory, and, at best, lacks the convenience and elegance of installable, fetchable handlers.

Finally, we consider a class of problems which I call migration problems, difficulties inherent in making a single-user system safe for multiple simultaneous users. RT-11 has a simple directory structure and a strong device orientation. The RT-11 user owns all devices and files, and therefore can FORMAT, INITIALIZE, and SQUEEZE as desired. Multi-user systems need multiple directories and a strong file orientation because devices (like disks) are shared by several users; device operations like INITIALIZE and SQUEEZE can be disastrous to many users. Although TSX-Plus solves some of these problems through features like operator privilege and controlled access to handlers, carefully partitioning disks into user-private subdevices and limiting user access to structured physical devices are essential to a more complete solution.

USER-ORIENTED EXTENSIONS

It is in the area of user extensions that TSX-Plus truly shines. Its unique features, such as transparent spooling, virtual lines, and detached jobs, enhance productivity even in the single user environment. To permit users to control these features, TSX-Plus includes many new commands and options.

These features and associated commands can make the transition between TSX-Plus and RT-11 very difficult, especially for unsophisticated users. As a simple example, KED accepts control-W as a command to repaint the screen. Under TSX-Plus, control-W signals a possible transition to a virtual line; thus, two consecutive control-Ws must be entered to repaint the screen.

Beyond supporting unique commands and options, TSX-Plus permits keyword abbreviations in places where RT-11 does not. RT-11 treats all SET commands as handler SET commands; option keywords cannot be abbreviated. TSX-Plus treats non-handler SET commands as keyboard commands, permitting options to be abbreviated. For example, SET TT NOQUIET and SET EDIT K52 may be abbreviated as SET TT NOQ and SET EDIT K5, respectively. If you are lazy (like most of us), routinely use these abbreviations, and place them in command files, the command files will not work under RT-11.

One of the more powerful features of TSX-Plus also causes severe reverse compatibility problems: command file extensions. These unique features, parameter substitution, special commands, and control characters, make command files unacceptable to RT-11. With a few important exceptions, all TSX-Plus command file extensions can be compatibly implemented under IND.

TSX-Plus can pass up to six parameters to a command file when it is invoked; these parameters may be substituted anywhere in command lines as desired. IND provides a similar capability allowing up to nine parameters to be passed.

TSX-Plus command files support two unique commands: DISPLAY and PAUSE. DISPLAY outputs a string (which may contain control characters and escape sequences) to the user terminal then continues command file execution. PAUSE displays a string then suspends command file execution pending user entry of carriage return. The functionality of both commands is provided by IND external comment lines and .ASK directives.

Control characters inserted in TSX-Plus command files regulate command line echo and terminal input/output. For example, ^(inhibits and ^) enables display of command file lines as they are executed, like the IND .ENABLE/.DISABLE QUIET commands. ^! suppresses all terminal output, including command line echo and program output. ^> causes all terminal input to be taken from the command file, while ^< restores normal terminal/command file input rules. None of the last three sequences has an equivalent in IND or RT-11; their inclusion would, however, be a welcome addition.

USER-ORIENTED GOTCHAS

With several minor and one major exception, common portions of RT-11 and TSX-Plus user interfaces behave identically. Most users, even those with no computer experience other than RT-11, are able to move freely and safely between the two provided some care has been taken in partitioning and assigning disk storage.

Many minor gotchas reflect the fact that TSX-Plus does not "look like" RT-11. Character echo is (normally) deferred, command lines can echo in lower case, and there is no extra blank line between monitor (dot) prompts. SHOW commands and their displays are very dissimilar. Most users adjust to these cosmetic differences quite readily.

A more serious potential problem is keyboard command abbreviation changes caused by unique TSX-Plus commands. Although TSX-Plus has many unique commands, keywords were obviously chosen carefully to prevent conflicts with RT-11 commands. Two exceptions were discovered: under TSX-Plus, DISMOUNT cannot be abbreviated DIS because of the DISPLAY command, and SET cannot be abbreviated SE because of the SEND command. Although neither of these is likely to cause undue hardship, future RT-11 or TSX-Plus command additions may adversely impact current command files and user habits.

Special hooks for COBOL-Plus, another S & H proprietary product, have also caused problems. If the first source file in a COMPILE command has extension .CBL, TSX-Plus invokes the COBOL-Plus compiler. Worse, if the first object file in a LINK command has extension .CBJ, TSX-Plus invokes the COBOL-Plus linker rather than LINK. This can be annoying, for example, to users for whom .CBJ means C-Object rather than COBOL-Object.

The most serious user-oriented gotcha involves command input interpretation rules. Under TSX-Plus, the following rules are applied:

1. If the command begins with @, the named command file is executed as an indirect file if KMON is set NOIND or as an IND control file if KMON is set IND. If the file does not exist, an error occurs.

2. If the command begins with $@, the named command file is executed as an indirect file. If the file does not exist, an error occurs.

3. If the command begins with #@, the named command file is executed as an IND control file. If the file does not exist, an error occurs.

4. If the command keyword is a keyboard command abbreviation, the command is executed; otherwise, perform step 5.

5. If a command file (.COM) with the same name as the command keyword is found on DK:, the file is executed as an indirect file if KMON is set NOIND or as an IND control file if KMON is set IND; otherwise, perform step 6.

6. If a command file (.COM) with the same name as the command keyword is found on SY:, the file is executed as an indirect file if KMON is set NOIND or as an IND control file if KMON is set IND; otherwise, perform step 7.

7. If an executable file (.SAV) with the same name as the command keyword is found on SY:, the file is run by simulating an R command. Parameters on the command line are passed to the program. Otherwise, perform step 8.

8. If User Command Linkage (UCL) is enabled and a file named UCL.SAV exists on SY:, the file is executed. The entire command line is passed to UCL.SAV. Otherwise, display an error message.

Under RT-11, command evaluation is very different. Steps 1, 2, 4, and 8 are the same as TSX-Plus. Steps 3, 5, and 6 are not performed by RT-11 (if step 4 fails, RT-11 next tries step 7). RT-11 performs step 7 by simulating RUN SY: rather than R (there is a world of difference between R and RUN under RT-11). As an example, assume the existence of XYZ.COM and XYZ.SAV on SY:. In response to the command XYZ, RT-11 executes SY:XYZ.SAV, but TSX-Plus executes SY:XYZ.COM as either an indirect or IND control file. To preserve compatibility, use explicit forms, like @SY:XYZ, R XYZ, and RUN SY:XYZ; use R IND followed by a command line rather than the #@ of step 3.

PROGRAMMER-ORIENTED SHORTCOMINGS

The TSX-Plus programming environment is nearly identical to that of RT-11. The few missing elements affect relatively few programmers and applications. For example, TSX-Plus does not permit more than 16 channels to be defined; RT-11 allows up to 255. Queue elements are allocated internally by TSX-Plus, which ignores attempts to increase the number of queues via .QSET. The number of fork blocks is determined by the NUMFRK parameter in TSGEN.MAC and cannot be increased by allocating fork blocks within handlers. Users who experience repeated "out of fork blocks" system errors may increase the NUMFRK parameter even though it is outside the user-modifiable portion of TSGEN.MAC.

TSX-Plus fails to support RT-11 interjob communication via .SDAT/.RCVD programmed requests or the MQ handler, using instead a unique message channel facility which must be enabled during SYSGEN.

The most serious TSX-Plus shortcomings are those related to real-time processing. Many RT-11 programmed requests, such as .PROTECT and .UNPROTECT, are missing from TSX-Plus altogether; others, including .FORK, .INTEN, .MTPS, and .SYNCH, may be used only in handlers. Unique TSX-Plus system calls provide a basic but not very responsive level of real-time support. The author strongly suggests incorporating real-time activities into a device handler to achieve faster response and to maintain compatibility with RT-11.

PROGRAMMER-ORIENTED EXTENSIONS

Most of the programmer-oriented extensions, largely new programmed requests, provide program access to the user extensions described earlier. Several requests control file sharing among jobs, necessary for programs like data base management systems which update files in place. Others control the unique real-time and interjob communication facilities of TSX-Plus and so replace the corresponding RT-11 requests.

TSX-Plus provides several requests which facilitate terminal input/output operations; these would be welcome additions to RT-11. Other requests control performance monitoring and the use of shared run time systems.

To maintain compatibility with RT-11, programs should ascertain that they are executing under TSX-Plus before using any of these unique requests; this procedure is detailed in TSX-Plus documentation. For example, a data base management system should not mark files as sharable or lock blocks within a file unless execution is under TSX-Plus; otherwise, fatal errors will occur. One compatible technique sets a global flag during program initialization reflecting whether RT-11 or TSX-Plus is running. Key routines, such as block locking routines, interrogate the flag to determine which programmed requests should be issued. This approach provides complete compatibility with minimal overhead.

PROGRAMMER-ORIENTED GOTCHAS

Programmers find gotchas to be the most serious of the differences between RT-11 and TSX-Plus. Many applications are unaffected by these gotchas, but this is a smaller group than that mentioned in the discussion of shortcomings.

A major class of gotchas stems from the need for external help under TSX-Plus to effect what programs alone can do under RT-11. For example, RT-11 enables single character terminal input (special input mode) whenever a program sets bit 12 in the Job Status Word (JSW) and issues a .RCTRLO programmed request. Under TSX-Plus, the program must perform the same actions but these are no longer sufficient; in addition, the program must send a special output sequence to the terminal, the user must append the /SINGLECHAR switch to the run command, or the user must issue a SET TT SINGLE command before running the program. Similar circumstances surround nowait terminal input (bit 6 in the JSW).

Use of .SPND and .RSUM requests also requires outside intervention. In this instance, the host system must have included real-time support in its TSX-Plus SYSGEN. Otherwise, .SPND and .RSUM are ignored, the application becomes processor bound, and overall system performance is severely degraded.

Programs which directly access the I/O page are even more adversely affected. Not only must the host system include real-time support, but the program must be run on a (time-sharing) line with operator privilege, and it must issue a programmed request to map the I/O page before attempting access. Otherwise, access is to the data structures of a simulated RMON which occupies the first several hundred bytes of the I/O page virtual address space.

The .PEEK and .POKE requests also behave differently. Under TSX-Plus, addresses in the range 160000 through 160612 reference the simulated RMON rather than the I/O page as they do in RT-11. Unfortunately, this range includes part of the floating CSR region for both UNIBUS and QBUS machines where special devices like DZV11s and IBV11s are frequently located. The TSX-Plus workaround is to map the I/O page and make direct access, the very practice .PEEK and .POKE were intended to eliminate!

TSX-Plus handlers cannot unilaterally determine their own job abort procedures. Under RT-11 (FB and XM), flags and handler processing determine how queues belonging to aborting jobs are disposed of. Under TSX-Plus, however, handler abort code is executed only if a SYSGEN flag is set or a user with operator privilege enables this procedure.

TSX-Plus handlers differ from their RT-11 counterparts in other ways as well. One major difference is the Page Address Register (PAR) through which the user buffer is mapped. RT-11 XM, loaded into high virtual address areas, uses PAR 1, while TSX-Plus, loaded into low virtual address areas, uses PAR 6. This difference is significant only if the handler accesses the user buffer directly or performs page bound checks on the buffer address. TSX-Plus versions of $GETBYT, $PUTBYT, $PUTWRD, and $P1EXT automatically compensate for variations in PAR usage.

Another difference in handlers is the job number field (Q.JNUM) in I/O queue elements. RT-11 stores the job number, always even, in bits 11-14 of word 3 of the queue element; bit 15 is reserved. TSX-Plus stores the job number in bits 11-15 of the same word; the number may be odd or even. In RT-11, the job number of the active queue element should be used when initializing a handler timer queue element. Under TSX-Plus, better performance can be achieved using a job number of 0 to eliminate synchronization of the completion routine with a (possibly swapped) user job.

To maintain compatibility between RT-11 and TSX-Plus handlers, the author defines a handler flag like MMG$T called TSX$P. In TSX-Plus handlers, the flag is set to 1; otherwise, it is 0. Conditional assembly of handler code based on this flag accomodates differences in PAR usage, queue element formats, and other idiosyncratic quirks.

SUMMARY

The author has been involved with RT-11 since Version 2C and with TSX-Plus since version 1.5. At one time, a discussion of compatibility issues would have required a paper ten times as long as this. Lately, the two systems have become more similar, which pleases and helps those of us who write software for both environments. The increased cooperation evident between DEC and S & H should help preserve the gains that have been made.

TSX-Plus is definitely a lot like RT-11; it is not hard to write most software to run under both, nor is it difficult to train people to use both without confusion. Neither of these goals is automatic. Differences exist now and other differences will exist in the future, but three general rules will probably always apply:

1. Most applications running under RT-11 will also run under TSX-Plus; a TSX-Plus SYSGEN or different run command may be needed, but the program will work.

2. Many RT-11 XM handlers will also run under TSX-Plus. Write new handlers with TSX-Plus in mind (use TSX$P) and use conditional assembly features freely; transportability is worth it.

3. No real-time applications are compatible. At the risk of being repetitious: if you want responsive and/or compatible real-time under TSX-Plus, do it in a handler!

A COMPARISON OF REAL TIME I/O USING RT-11 AND TSX-PLUS

Peter N. Johnson
National Center for Atmospheric Research*
Boulder, Colorado

ABSTRACT

A comparison is made between the real time I/O capabilities of RT-11 and TSX-Plus for a specific application that required input of a block of 26 words during a 40 msec. time slot each 500 msec. The hardware included a DRV11 parallel interface and LSI-11/23 processor. The application was implemented with both inline interrupt code as well as a device handler under both RT-11 and TSX-Plus. The various time slice parameters of TSX-Plus were used to adjust the system response for this application. The effects on interrupt response of each of the various combinations of interrupt handling and operating systems are discussed.

1.0 INTRODUCTION

A real-time application often requires responding to interrupts under some time constraints. The choice in how to deal with interrupts is between two approaches — inline interrupt service routines and device handlers. In this study, issues relating to these two choices are discussed and the results of implementing both of them are compared using both RT-11 and TSX-Plus.

First a brief description of the requirements is in order. The Multiple Aircraft Position System (MAPS) was developed at the National Center for Atmospheric Research in response to the need for aircraft position in the coordination of weather research field projects. Radio interferometer techniques are used to provide real-time positions for up to 10 aircraft. The system samples one aircraft at a time from each of three remote tracking sites. These data are then transmitted to a master control unit which formats and buffers the data for transfer to a computer. The computer then performs calculations to determine the aircraft position.

When this system was first tested in 1978, a floppy disk based PDP-11/03 running RT-11 was dedicated to this application. Data was input to the computer by an inline interrupt service routine. Oregon Pascal-1 was used in combination with a few lines of Macro-11 assembly language code to accomplish this. Gradually the computer system has been expanded to include hard disks, tape drives, LSI-11/23 and more memory. Eventually it became desirable to expand the use of this more powerfull computer system to include other applications because it was no longer economical to continue dedicating the computer system to the MAPS application. It was felt that a multi-user environment was more suitable to the needs than the Foreground/Background environment provided by RT-11. Since compatibility with RT-11 was desirable, the decision was made to use TSX-Plus.

In addition to the requirement to input the MAPS data, there are several output channels involved in distributing and displaying the aircraft position information. Of course the normal computer console and printer facilities are used as well as a mass storage device for recording the data. Also four output channels were used to service serial data links to various display devices. These output requirements were satisfied using fairly simple device handlers with the serial interfaces.

The MAPS master control unit was interfaced to the computer system by a DRV11 parallel interface. The data was transfered as 16 bit parallel words. Since the system samples an aircraft position once every 500 milliseconds, there is a new block of 26 words available every 500 milliseconds. When the master control unit has the data ready, the REQB line of the DRV11 parallel interface is asserted. The computer must respond to this signal and complete the data transfer within 40 milliseconds. Any data that is not transfered by the end of this period is lost.

In general there are three strategies to consider when responding to external events -- programed I/O, inline interrupt service routines, and device handlers. First, since there are several asynchronous events to deal with in the main application program, it was considered inappropriate to use programmed I/O. Thus only the two strategies using interrupts were considered for this application. The next issue was how much code should be

*The National Center for Atmospheric Research is sponsored by the National Science Foundation.

activated in response to the interrupt. Because of the multiplicity of devices using interrupts on the system, it seemed desirable to minimize the code that was executed in response to the interrupt. Most of the data calculations and operations are performed in a main program and the interrupt response is limited to actually transfering the data into a buffer and somehow providing a signal to the main program that the operation is complete. In the following sections, the techniques used to interface the interrupt response with the main program are discussed.

2.0 INLINE INTERRUPT SERVICE ROUTINES

An interrupt service routine is a section of code that is executed in response to an interrupt. In the case of this application, the interrupt service routine was responsible for reading in a new block of data through the DRV11 interface and passing the results to the main program. Using an inline interrupt service routine allowed this process to be accomplished with a minimum of system overhead under RT-11. A set of buffers and flags are used to communicate with the main program. Actually there were several buffers along with appropriate pointers that were used to transfer data to the main program. The multiplicity of buffers allows the system some flexibility in executing the main program. This was especially useful in allowing the interrupt driven input to proceed when the main program execution was temporarily delayed.

The flags are used to indicate to the main program when new data are available and such conditions as data overflow or errors. In general the flags are set by the interrupt service routine and cleared by the main program. When the data transfer is complete, the inline interrupt service routine sets a flag to indicate that there is new data available. Upon return from the interrupt service, the rest of the program detects that new data is available and performs the appropriate operations on the data. When the operations are complete, the program clears the flag and waits for a change in the flag to indicate more data are available. The process of waiting for a change in the flag involves a loop that continuously checks the state of the flag.

Since the original application used an inline interrupt service routine it was natural to modify the existing code to function with the real-time program support available under TSX-Plus. The main modification that was necessary was to allow access to the DRV11 registers in the I/O page. It was also necessary to select a priority at which the completion routine would execute. For lack of a better argument, the completion routine priority was chosen to be 7 to minimize the chance that the inline interrupt service routine would be interrupted long enough to prevent completing the transfer of data within the 40 millisecond time limit. Note, however, that only the short section of code that actually transfers the data runs at completion routing priority. Since the I/O page mapping and the completion routine priority selection involve system overhead that does not exist under RT-11, this approach may be expected to have slower response under TSX-Plus.

Upon return to the main program, operation proceeds as a compute bound job and competes with other jobs for CPU resources. As mentioned the buffering is designed with room for several blocks of data so that no data is lost if the next data is available before processing is completed. Of course, processing must keep up with the data rate on the average or some data will eventually be lost. The MAPS application requires about 30% of the CPU resources under TSX-Plus. Thus there are several features that are designed to facilitate sharing CPU resources with other jobs. The MAPS application was run with a combination of compute bound and I/O intensive jobs as seemed to have enough priority to get the CPU resources required to keep up with the real time data. The various system parameters, e.g. QUAN1 or QUAN1A, did not seem to have any effect on this.

3.0 DEVICE HANDLERS

Device handlers are interrupt service routines that provide a standardized interface to the devices that are part of the computer system. The advantages of I/O using device handlers are device independence, simplicity of use and sharing of CPU resources with other processes. A certain amount of system overhead is required for these standard interfaces. In this application, it was of some interest to see how the use of a device handler would affect the sharing of CPU resources.

RT-11 and TSX-Plus provide three types of I/O using device handlers -- synchronous, asynchronous, and event driven I/O. Of these, asynchronous I/O is of most interest in this application since processing is allowed to proceed in parallel with I/O activity. However asynchronous I/O is not part of the standard Pascal file capability. This was overcome by developing a file handling library that was flexible enough to take advantage of all the I/O programmed requests of RT-11 and TSX-Plus.

As with inline interrupt service routines, it was desirable allow data processing to temporarily fall behind the data input. Using device handlers and the standard system I/O calls, this was easily accomplished by queueing several requests. Thus if the main program does not complete processing a block of data before the next one comes along, the system simply uses the next input request on the I/O queue. There is a flag that is part of the queued buffer that informs the main program that a particular input request has been satisfied.

The priority of the actual input process is not an issue as it was with the inline code because the device handler executes at system level and only other system tasks compete with it. Upon return to the main program after completion of the input process, the application processing proceeds at I/O completion priority. The time spent at this priority level is controlled by the TSX-Plus system parameter QUAN1A. If QUAN1A is set to a time that is long enough to complete all the data processing, then the application will be have enough CPU resources to run properly. As long as there are not too many other jobs receiving the same or higher priority, the main program will not fall behind. Under TSX-Plus, the main program required about 30% of the CPU run time which translated to 150 milliseconds every 500 milliseconds. Thus a setting of 2, which allows 200 milliseconds, for QUAN1A was sufficient to allow the application to keep up with the real time data.

4.0 DISCUSSION

As is the case with any time-sharing system, the best solution for a real-time application must take into account the speed of response, programming difficulty, and effect on sharing the system resources. The response of the computer system to the real time data was measured by connecting a scope to the REQB line the DRV11 interface. In this way both the interrupt response time as well the the complete data transfer time was directly observable. The interrupt response time is defined as the time interval from data becoming available (as signaled by the first assertion of REQB) and the first word read (as signaled by the first negation of REQB). The data transfer time is defined as the time interval from data becoming available to the last word read. Table 1 shows the results of the measurement of data transfer time.

	RT-11	TSX-Plus
inline	1.9	5.2
handler	1.8	2.2

Table 1. Data transfer time (in milliseconds).

Note that all the times are comparable except that inline code under TSX-Plus is substantially longer. This could be either because of interrupt response or the time to transfer each word. A look at Table 2 shows that the difference is due primarily to the interrupt response of the inline code under TSX-Plus. Apparently, the TSX-Plus system overhead to pass an interrupt on to the inline code is substantially more that RT-11. Note, however, that even this longer time is still short enough to complete the data transfer within the required 40 milliseconds. Nevertheless the handler has the advantage in speed of response.

	RT-11	TSX-Plus
inline	0.11	3.3
handler	0.05	0.1

Table 2. Interrupt response time (in milliseconds).

In this application, the programming difficulty was comparable for the inline code and the handler. In the case of the inline code there was some added complexity for the modifications required for TSX-Plus. In the case of the handler, there was no difference between the RT-11 application and the TSX-Plus application except for the usual conditional assembly sections in a handler intended for use with memory management. Overall the handler seemed more suitable because the main program was more concise and easier to program and read.

Of more interest is the effect on other users. In this case there did not seem to be a great deal of difference except perhaps in the system memory allocation for the device handler required by the TSX-Plus system. The permanent memory allocation required for the handler does reduce the total memory available to share among other users but not significantly since the handler takes less than 500 bytes. In general, the application took about the same resources (30% CPU time) to execute for both inline code and handler code under TSX-Plus. The application ran somewhat faster under RT-11. As long as QUANIA was set properly, it seemed that other users had reasonable access to the computer resources using either approach under TSX-Plus. This is not a factor under RT-11, of course. Taking all these considerations into account the conclusion reached is that for this application the device handler was the preferred solution.

It is interesting to compare these results with the article that appeared in the May 1983 issue of Mini-tasker entitled "TSX-Plus Real Time Support Facility" by John Yardley. It appears that the preference expressed above for handlers is consistent with the conclusions reached in the article. A point of disagreement is the response time for interrupts using inline code. The article stated that it could be as long as 2 mS, while these measurements indicate that it could be somewhat longer. This certainly supports the conclusion than TSX-Plus real-time completion routines cannot handle interrupts at much more than 100 per second.

TSX-PLUS INTERNALS

Jan Bramlet
S&H Computer Systems, Inc.
Nashville, TN 37212

ABSTRACT

TSX-Plus uses various optimization techniques and data structures to enhance performance. The presentation will cover the internal structure and organization of TSX-Plus with special emphasis on memory usage, data structures, and system values associated with performance.

INTRODUCTION

Since every computer site is unique, there does not exist a single "best" TSX-Plus system generation selection. Performance depends on both the system configuration and its actual use. In order to tune system response, the application programs must be analysed. Together with the knowledge of their features and a basic understanding of the operating system internals, decisions can be made to improve system performance. System tuning is an on-going process which becomes more obvious with increased experience.

There are three areas related to system performance tuning: memory organization, I/O optimization, and job execution scheduling. In order to improve one area, a sacrifice must usually be made in another.

MEMORY ORGANIZATION

Memory organization is divided into two areas: memory used by the operating system and memory used by user's programs. The memory used by the operating system is permanently allocated and contains code regions and data structures reserved for its exclusive use. The user's memory is generally swappable (depending on system generation), pooled and allocated dynamically. Along with each job image, a 3 K byte job context region is also allocated.

The operating system is divided into four distinct regions. The kernel root code begins at physical memory address zero. It is mapped using kernel PARs (page address registers) 0 through 4 which are static and therefore dedicated mapping registers. Because of this, the kernel root code region is restricted to a maximum of 40 K bytes. The I/O page is the only other static region and is mapped using kernel PAR 7.

The mapped data regions reside directly above the kernel root code in physical memory. Each data region is an individual data storage area which is mapped using kernel PAR 6. Because of this, each data region is bounded to the nearest 64 byte address and restricted in size to a maximum of 8 K bytes. Only one data region may be accessed at a time.

The memory resident overlay code regions are loaded starting at the top of physical usable memory. Each overlay code region is mapped using kernel PAR 5 and therefore is bounded to the nearest 64 byte address and restricted in size to a maximum of 8 K bytes. Only one memory resident overlay code region may be mapped at a time.

When executing, TSX-Plus resembles the following virtual address layout:

TSX-Plus Use of Virtual Memory

```
+---------------------------+ 177777
|          I/O Page         |
+---------------------------+ 160000
|  Mapped Data Regions      |
+---------------------------+ 140000
|  System Overlay Regions   |
+---------------------------+ 120000
|                           |
|                           |
|     Kernel Root Code      |
|     and Data Region       |
|                           |
|                           |
+---------------------------+ 0
```

The kernel root code contains device handler vectors (located from zero to octal 500); the memory resident overlay handler and tables necessary for interfacing to overlay code sections; TSGEN generated data tables; executive code consisting of the job swapper, scheduler, etc.; optional features code (inter-program message communication, record locking, and spooling); I/O related processing code; clock and terminal interrupt entry code; and the startup initialization code. To conserve space, TSX-Plus allocates data structures which do not require initialization on top of the startup initialization code. If additional space is necessary, the top of TSX-Plus will be extended. These buffers consist of the job information tables (simulated RMON); I/O queue elements; system message buffers; directory cache buffers; and spooling buffers. The device handlers are then loaded above these data buffers. Finally, the memory map is allocated. The entire kernel root region size must not exceed 40 K bytes.

The mapped data regions allocated during startup consist of the terminal input and output character buffers; the data cache buffers; and the performance monitor buffers. Each data buffer is individually accessed.

Currently there are six memory resident overlay code regions. They are logically separated by function which naturally reduces the number of calls necessary to the overlay handler since only one overlay code region may be mapped at a time. The overlay code regions perform (1) terminal input and output requests; (2) programmed EMT requests; (3) program logical address space requests (PLAS); (4) directory manipulation requests (USR); (5) real-time service requests; and (6) miscellaneous executive functions such as fatal error processing.

Thus, TSX-Plus grows upward from a physical base address of zero and downward from the top of usable memory. A fifth region which the operating system pre-allocates as dedicated memory but does not directly use, is user defined shared run-time systems. It is positioned directly below the memory resident overlay code regions.

When executing, TSX-Plus resembles the following physical memory layout:

TSX-Plus Use of Physical Memory

```
+-----------------------------+
|          I/O Page           |
+-----------------------------+
-                             -

-                             -
+-----------------------------+  Top of memory
|   System Overlay Regions    |
+-----------------------------+
|      Shared Run-Times       |
|         (optional)          |
+-----------------------------+
|                             |
|                             |
-                             -
        User's Job Region
-                             -
|                             |
|                             |
+-----------------------------+
| Performance Monitor Buffer  |
|         (optional)          |
+-----------------------------+
|     Terminal I/O Buffers    |
+-----------------------------+
|     Data Cache Buffers      |
|         (optional)          |
+-----------------------------+  Maximum 40Kb
| Memory Map Table            |
| Device Handlers             |
| Initialization Code         |
| (Optional Features Code)    |
| Executive Code              |
| TSGEN                       |
| Overlay Tables              |
| Interrupt Vectors           |
+-----------------------------+  Physical 0
```

The user's job region is the contiguous memory sandwiched between memory used by the operating system. It is allocated dynamically, placing each user's job in the first available free memory area large enough to fit. In a swapping system, the job can potentially be positioned anywhere within the region. A 3 K byte job context region remains appended directly below the job image. This allows both the job image and its associated context region to be swapped as a single entity.

The user's virtual address space is limited to 64 K bytes. The program may request and be granted more physical space by use of the PLAS requests. The memory regions allocated need not be contiguous with the job's original program image. When the job is swapped, the PLAS regions are swapped into a separate disk file. A program may also remap its normal program image through the use of real-time or shared run-time EMTs.

I/O OPTIMIZATION

TSX-Plus uses I/O internally for spooling, job swapping, and directory operations. Up to three job swap files and one spool file can be created during initialization. TSX-Plus retains the position of these files which restricts their movement but eliminates the necessity for any additional directory operations. Likewise, the position of the concise command language interpreter (CCL) and the keyboard monitor (TSKMON) are only located during startup.

Spooling is the method of placing output to slow devices on a mass storage device to await further processing. When an I/O request is queued to a device handler the job is temporarily locked into its current memory position. This is necessary since the handler will actually perform the data transfer by using the positional information located in the I/O queue element. Since the spooling system intercepts the I/O request and transfers the data into its own spool buffer, the job will return to an executable state and become eligible for swapping much sooner. In addition, TSX-Plus will always attempt to double buffer the spooled output request if two or more buffers have been defined.

The directory cache is a memory resident cache of file directory entries maintained for speed in opening existing files. If a requested file entry is found within the directory cache, the file will be opened without the necessity of any disk I/O. The directory cache stores the starting directory block position, physical device name, file name and extension along with the starting block number and length. The file entries in the cache are maintained in most recently used order. When a file not in cache is opened, the least recently used entry is replaced with that file entry. Operations involving directory alterations (such as enter, delete, and rename) affect the physical disk as well as the memory cache; therefore, no speed increase is realized. Directory caching is automatic only for the system device but can be initiated for other devices by use of the "MOUNT" keyboard command. When exchanging disks, caution should be observed to insure that the device is dismounted from directory caching before accessing the new disk.

The data cache is a memory resident cache of data blocks maintained for speed in reading data files. A special programmed request must be issued to enable data caching after a file has been previously opened. If a read is issued to a file eligible for data caching, the memory resident cache buffers are first searched to see if the data block is already contained in one of the data cache buffers. When the data block is found in memory, the data is moved to the user's buffer and no disk I/O is performed. If the data block is not in cache, it is read into a cache buffer and moved to the user's buffer. The operating system always performs a write through the cache, altering both cache and the physical device so that file corruption will be avoided. An efficient algorithm is used to keep the most active blocks in memory. Each block is weighted with an activity counter and the least active block is deallocated when a new block is read. This has the advantage of maximizing speed for ISAM structured files.

EXECUTION SCHEDULING

TSX-Plus uses a dynamic priority system for job execution scheduling. Each job can be classified into one of four general state categories: real-time interrupt service, high-priority, CPU-bound, and waiting for event or resource. These categories are further subdivided into smaller classes.

The real-time interrupt service facility has two subdivisions: a real-time interrupt service routine and a real-time interrupt completion routine. A real-time interrupt service routine can run without requiring a scheduling call by running at fork level in user mode but without the full context of the job. This method restricts the allowed executable EMTs within the interrupt service facility but provides fast interrupt response. Since the real-time interrupt service routine is executed at fork level, each is queued and executed in order of occurrence and intermixed with system interrupt service fork routines.

A real-time interrupt completion routine does require a job scheduling cycle which will perform a context switch before entering the interrupt completion routine. There are seven (1 through 7) software priorities which are only used by the scheduler to determine which real-time completion routine will take precedence over others running or waiting to run. A real-time completion routine may run until it relinquishes control or is suspended by a higher priority interrupt service routine. However, if the real-time completion routine enters a wait state, when it resumes execution it then becomes scheduled as a normal job. An eighth software priority (zero) is available to classify the real-time completion routine in a high-priority state.

In addition to the zero priority real-time completion routine, the following states are high-priority states: completion of a timed wait (.TWAIT) or mark time (.MRKT); terminal input finished (activation character received); terminal output buffer empty; I/O completed; and terminal output buffer low. There is only one CPU-bound state.

Currently there are fourteen states to identify jobs waiting for events or resources. Obviously, these states are not executable. Since each of these job states is unique, when a resource becomes available, those jobs waiting are easily identified and scheduled for execution.

Each of the job states has a unique number which numerically reflects its relationship to the other job states. Every job, whether executable or in a wait state, will have one of these unique numbers defining its current status.

Recently, a priority structure has been added to TSX-Plus which allows the definition of a user specified job priority number between 1 and 99. The priority value given a job reflects is relationship to other jobs with identical job states. When a job is disconnected from the current line by switching to a virtual line, the job's state remains the same but the job's priority is decremented by the amount specified for the system generation parameter named VIRPRI.

Three time-slice parameters control the execution of the high-priority and CPU-bound states. QUAN1 controls the maximum length of time a job will execute following completion of a timed wait or a terminal input request. QUAN1A controls the maximum length of time a job will execute following a terminal output buffer empty, I/O completed, or terminal output buffer low state. Should a high-priority job execute its full time-slice (either QUAN1 or QUAN1A) without entering a wait state, it will automatically become a CPU-bound job. QUAN2 is the maximum length of time a job will execute while in a CPU-bound state.

A new system generation parameter named HIPRCT controls the execution state of I/O intensive jobs. HIPRCT controls the number of consecutive times a job will be placed in a high-priority state following an I/O completion event. When the job has exhausted this count, it is placed in the CPU-bound state and its count is reset to HIPRCT.

The job queue contains a list of all jobs, ordered first by job state number then within job state number by the defined priority number. Then the role of the scheduler becomes very simple.

1. Obtain the job at the head of the executable queue.

2. If the job is not currently in memory, then obtain the program image from the swap file. It may be necessary to swap lower priority jobs from the tail of the job queue.

3. Run the job until it either enters a wait state, the allotted time-slice has expired, or a higher priority job becomes executable.

   a. If the job enters a wait state, it is removed from its current queue position and is placed in the queue in the appropriate wait state.

   b. If the allotted time-slice has expired, the job is considered to be CPU-bound. It is removed from its current queue position and placed in the queue at the tail of the CPU-bound jobs with respect to its defined priority. Note that its original state could have been either high-priority or CPU-bound.

   c. External events may interrupt the executing job before it either enters a wait state or its time-slice expires. The interrupted job remains in its current queue position (the higher priority job has been queued in front of this job) and when resumed will execute for the remainder of its time-slice.

A timer is cleared when a job is brought into memory from the swap file. As long as the job remains executable, it will not be eligible for swapping out of memory until the timer reaches the time quantity specified by the CORTIM parameter. The job becomes immediately eligible for swapping if it enters a wait state other than waiting for completion of a non-terminal I/O request.

Jobs may also remain in memory for other reasons. A job which uses the real-time EMT to lock itself into memory will not be swapped. Also, a job issuing a non-terminal I/O request is temporarily blocked from swapping once the request is queued to the appropriate device handler. It must remain in memory until the I/O operation completes. A flag is maintained for purposes of "holding" I/O from the device handler queue if the job has already used its time-slice.

TUNING SYSTEM PERFORMANCE

Since the essential duty of the operating system is to execute user's programs, the most important tool in tuning system performance is knowledge of the features utilized by these programs. How much total memory is available? What is the normal size of the typical program? Is a shared run-time system available? Which features do the application programs use - inter-program message communication, record locking, data caching, real-time EMTs, etc?

To optimize memory, generate reasonable values for system parameters. Since the operating system's image is permanently resident, space should not be wasted for features that will not be used. Use a shared run-time system for reducing user's memory requirements, but remember, since this will be permanently resident, at least three or more users should frequently require this run-time. Use the SYSMON utility and the "SHOW MEMORY" command to obtain information about memory usage for the operating system and the user's programs. Use the "MEMORY" command to dynamically tune particular programs and the SETSIZ utility once the optimum memory size is obtained. Calculate a reasonable memory default partition for executing most user programs and use the SETSIZ utility to increase space for those programs which will not fit.

To optimize I/O, allocate reasonable user memory partitions in order to keep as many jobs in memory as possible thus reducing swap I/O. Use spooling for slow speed output devices. This allows the job to become executable and eligible for swapping much sooner. Use directory cache and data cache when possible. This reduces disk head movement and eliminates reads when the item is located in the memory cache. Use a shared run-time system for reducing program overlay reads.

To optimize execution scheduling, QUAN1 and QUAN1A should ideally be set so that the majority of high-priority jobs will complete execution and enter a wait state (either I/O wait or terminal I/O) before the full time-slice expires. HIPRCT should be set such that an I/O intensive job will not totally dominate the system by remaining in a high-priority state. CPU-bound jobs will only execute if no higher priority job requires service and QUAN2 should allow them a long enough execution time to justify their scheduling and memory swap. CORTIM should be set high enough so that programs will not swap when the event they are waiting on will complete quickly; but, it should not be set so high that jobs which could soak up idle time cannot be swapped into memory. Execution of jobs may be fine tuned within each state by adjusting the associated priorities.

Finally, some attention should be given to the application programs. The performance monitor can be used for isolating program areas which require more execution time. By optimizing these areas, program performance should improve. The use of single character input should be avoided since this repeatedly places the program in a high-priority terminal input completed state. The use of no-wait character input degrades system performance even more since this places the program in a high-priority terminal input completed state without the necessity of an entered character. If at all possible, a line of input should be processed by entering characters followed by an activation character (this is normally a carriage return although other activation characters may be defined). During buffered input, the job is suspended and may even be swapped to disk to allow other jobs to execute. In addition, the use of high efficiency terminal I/O can reduce the overhead associated with terminal I/O.