RSX

MULTI-TASKER

AUGUST 1984 ISSUE

Printed in the U.S.A.

RSX MULTI-TASKER

TABLE OF CONTENTS

From The Editors

Submissions are dwindling again. No one wants to spend too much time at work during the lazy days of summer; therefore, I am hopeful submissions will pick up during the Fall. Many people are doing many interesting things with RSX. Please step into the limelight and let us all know what your doing. We can accept submissions in many machine readable forms. My address is below, please contact me if you need any assistance.

What's Inside

This month the North Texas LUG gives us a view of what they perceive to be the future via there Crystal Ball. The RSX SIG is planning three very interesting pre-symposium seminars for the Anaheim Symposium. Hans Jung, RSX SIG Pre-symposium Seminar Coordinator tells us what is in store. Several months ago a letter appeared in the Multitasker about putting a quote in a symbol within an RSX command file, this month we have two more solutions to this problem.

Three very interesting articles are presented this month. Philip Miller and Robert Novas discuss Serializing I/O in RSX-11M. Mark Chatterton brings to light the FMS-11 interface in the RSX command File Processor. I'm sure most of you have heard about Virtual Disks. This month Jim McGlinchey gives us some Spiffy ideas on practical uses for them.

Dominic DiNollo
Loral Electronic Systems
Engineering Computer System
Ridge Hill
Yonkers, New York 10710

(914) 968-2500 ext. 2210

THE CRYSTAL BALL

From the North Texas LUG Newsletter

The night is dark. In the distance, Coyotes hauntingly howl. The wind moans through the cracks in the house. Inside, the grou sits around a table. A solitary candle lights the room castin ghostlike shadows that dance about the room. Next to the candl lies a crystal ball. We all gaze intently into it. At first, it i shrounded in mist. But slowly, the mist parts to give us a view o the future.

Looking down into the murky depths, we see the planet Venu (from whence the mist came, no doubt). But the image of Venu slowly fades to reveal something called a 790...then, you alread know about that. But wait! There is more to know! It uses ne technology to gain speed at the expense of power consumption. There was something about a bunch of gates all in a row and colum but no further word came of that area. Performance...ah, yes That evasive quality that DEC doesn't talk about. Well, it doesn' stop the crystal ball! It knows nothing but tells all. I se floating in the mists a 4 MIPS figu... no! It says 9 MIPS... no it seems to be floating between these two values because of th highly pipelined design. And cost...yes, it's the price of tw 780s and the performance of four. Looking into the heart of th machine, there is something different, something strange. It sit over the boards...I can't quite make it out....it seems, yes, it' definitely is a 4 foot box fan positioned 1 inch above the board (for cooling, no doubt). It has a name on the side...let' see...ah yes, it says DECair. Mayhaps this is the source for som of the hot air coming from DEC on past occasions.

The image of Venus slowly fades to show...yes, yes, i is....new software. It is an operating system which (if you ar lucky enough to login) won't allow you to do anything due to al the checks and security features that have been implimented. Bu that's okay...you just wanted to play anyway. Management cares no that serious work can get done so long as it is the work is secur and safe in the friendly hands of VMS. Still, you feel goo knowing that your secure system has already been broken by severa intelligent users. But that is a contracdiction in terms intelligent and users...Ah, but such are the mysteries of th world.

I see a reduced instruction set computer...could this be the PDP-8 brought up to speed???

Ah, I see a book, no... more than one book... something called the M+ documentation set which makes references to a multi-processor system. Could such a device be forthcoming??? I see such a device being introduced within a year... The software already has embedded within it the necessary hooks and the hardware has been built and I see it in current operation on a software development system. What a system I see... speed of 4 11/73 boards all running M+ without the overhead of VMS; what ever that is. But then again, good ole stodgy VMS will still be there, proding slowly along, helping all you real time people lead happier, healthier lives.

The image fades....Something called a fully vectored executive of M/M+ slowly forms... This wonderous beast slowly gains strength and power, but dies a slow and painful death due to buyer apathy...I see forming something called named directories for M+ but not for M. And a hierachical directory structure. And logical name tables ala VMS. All this I see already working on both M and M+. And I see the implimentation done over one weekend...two days...such a short time for something that DEC has said would take years. And yet, no such product will come forth from the bowels of Digital...a monster it seems has run amonk within DEC...and it is exceedingly evil for it is the beast called...hush, don't say it too loudly...politics. And this beast has forbade new features on a product the DEC has "solidified". Such is the life for something which is supposed to be lean and mean.

The smoke slowly curls about a small creature called a Micro/RSX which has J-11 support, I and D space, supervisory mode support and even newer device support. The hushed conversations behind the CPUs have rumored performance 4 to 7 times that of the current Micro-11...and even a disk that's faster than a floppy!

I see a room somewhere in the dark corridors of power; a small elf, hoping to please the overlords, helps the poor memory starved M systems by shrinking the size of the powerful EDT weapon to a small shadow of itself, thus making it up to 10 times slower. But then to bedazzle the M+ cult, he makes it big and fast and yet, says not a word to the M clan about the fact that the unoverlayed M+ EDT will work just as well under M....and yet, not a word was forthcoming for help on EDT. Know ye well that help under EDT V3 "sorta" works...PF2 gives you the scrolling keypad layout...and type ye not a keypad less you find no help available...all this to reassure you that DEC has done a bangup job checking out the beast. Therefore, throw not out ye V2 EDTs lest the beast of V3 come up and smite thee.

I see that the lowely DMZ-32 slowly makes it's way to the sunlight to replace the powerful but hungary DMF-32...imagine! 24 ports on one board, all with full modem control and all with DMA.

I see DEUNAs coming complete with sabres and marshmellows sc that ye can have your picnic roasting the marshmellows over the 16 amp DEUNA.

And the terminal server (good help is so hard to find these days) that rivals DMF-32 performance and runs but on one coax wire...a wire different in diameter than the rest of the world's coax...but then again, you should only buy DEC anyway lest field service point its finger at the "other" vendors.

The images slowly fades into the mists as the first light of dawn treks slowly across the room. As we all look up, a booming, bodiless voice speaks the language of all people: "That will be thirty five dollars." As the echo fades slowly into the room, we are comforted by one thought that has given us strength throughout many, many years:

> "Though we may criticize DEC, though we may poke fun at them, though we curse them under, nay, over our breath, though busses may go and interconnects come, though users knoweth not what they what and DEC knoweth all, we still want our DEC machines and don't let anyone take them away from us."

SIG News

Recently several DECUS members who work with Data Acquistion and Real-Time Control requested the formation of a new SIG (acronym DAARC) that would cross operating system divisions and deal with the problems common to this type of operation regardless of which operating system is used.

The RSX SIG Steering Committee has voted in favor of the proposed formation.

Ed Cetron, the RSX Real-Time Working Group Chairman, recommended in favor of the new SIG. After some discussion among the Steering Committee members, we agreed that the new SIG would not be in conflict with the RSX SIG. People working with DAARC would still want to be active in the RSX SIG if they were using RSX.

At the moment, to the knowledge of this editor, the new SIG has not been formally chartered by DECUS.

RSX Pre-symposium Seminars in Anaheim

Hans Jung
RSX Pre-Symposium Chairman

Listed below are abstracts for the three RSX SIG presymposium seminars.

Public Domain Software: An Indepth Review of the RSX SIG Tapes

Some of the best programs available at any price are free: little programs that save a few minutes a day, but are too much trouble to write in house, or a larger package too expensive to buy and too big to write yourself. Someone, somewhere has written them, and they're in the public domain, but you have to know where to find them.

Just some of the programs available are:

- languages: BASIC, C, two flavors of PASCAL, RATFOR
- text processing packages: the TECO editor and RUNOFF formatting program
- Software Tools - a virtual operating system with UNIX-like tools
- CCL - a user extendable command line intrepreter to make RSX "friendly"
- Virtual Disks
- SRD - a directory utility
- KERMIT and other computer communications packages
- PortaCalc - a spreadsheet written in FORTRAN

Programs like these have been contributed by goverment labs, universities and private companies. This high quality software is distributed in the public domain through the RSX SIG tape copy proceedure.

A number of experienced RSX users will describe the programs available from seven years of RSX SIG tapes. In addition to the program description, tested methods for using SIG software in system management and program development will be covered. Using SIG programs as templates for custom software development will also be discussed.

VAX users should also be interested since many of the programs run in compatability mode, and some are even availible in native mode.

Anyone involved in systems or project management will come away with some immediately useful ideas for using these programs. The small programming shop, and particularly the one person programming staff, will find that someone has solved those nagging problems, at a price they can afford.

Notes on all the programs discussed will be provided. The programs discussed will be made availible on magnetic tape.

Speaker Biographies:

(Note - The seminar is being organized by Glenn Everhart and Jim Neeland, so their biographies are included. We expect to have at least 6 presenters, most from the Anaheim area.)

Glenn Everhart
Staff Engineering Scientist
RCA
Cherry Hill, NJ

Glenn Everhart has contributed numerous programs to the RSX SIG. His most notable current program is "PortaCalc", a spreadsheet written in FORTRAN which runs on both RSX and VMS. He is the current RSX SIG Tape Coordinator.

Jim Neeland
Hughes Research

Jim Neeland has been active in the RSX SIG tape copy program since its inception in 1977. He served for two years as RSX SIG Tape Copy Coordinator. Active in since 1975, he has contributed or worked on such popular RSX programs as TYPE, OPA (online pool analyser) and DVCDAT(device database display).

The other speakers will be announced at a later date.

RSX Systems Internals: A Developers View

RSX system internals have often been taught, but rarely by a person who actually wrote part of RSX. Brian McCarthy, an RSX developer, will describe not only how RSX works, but why the RSX group chose to make it work that way.

This full day overview of RSX should interest the novice systems programmer and the experience RSX "guru" alike. As important as the details of the internal structure, the philosophy

behind RSX will be explained.

Topics to be covered include memory mapping, the I/O structure, interrupts and traps, task scheduling and the data structures they use. With only a day to cover such a broad area, sources for further study will be suggested.

RSX-11M Plus specific areas such as secondary pool, I and D space mapping and supervisor mode will be described, both in what they are and why they were added to M-Plus.

Some experience with RSX systems programming will be assumed, as will a general knowledge of PDP-11 hardware. Those with systems experience on other operating systems and hardware may find this seminar an excellent introduction to the power of RSX for designing complex hardware and software systems.

Speaker Biographies:

Brian McCarthy
Principle Software Engineer
RSX Development Group
DEC

Brian McCarthy is responsible for a large portion of RSX 11M-Plus. He has worked with RSX for over nine years, first as an independent consultant. In 1977 he joined DEC working in Educational Services. He became a member of the RSX development group in 1979. His current projects include working with the J-11 and other new DEC processors.

Industrial Automation: Tieing your Real-time Device to RSX

Computers are great for talking to terminals and printers, but what happens when you want to talk to a numerically controlled lathe or get data from the fancy new instrument just delivered to your lab?

Interfacing to unusual peripherals is often learned ad hoc, with less than perfect results. Yet many techniques, from the very simple to the very sophisticated exist for integrating equipment into a real time system. This seminar will cover some of them.

Techniques covered will include mapping to the I/O page, using the connect-to-interrupt directive and custom written device drivers. All have advantages and disadvantages, and the tradeoffs will be described.

Designing a system structure to handle real time requirements is another critical aspect which will be covered in detail.

With many instruments now interfaced through EIA ports, the seminar will describe some of the special problems of using the terminal driver with non-terminal devices.

Case studies of actual systems will be presented as examples of how some real world problems have been handled.

The material will be covered from an RSX point of view, but many of the techniques described apply to other PDP-11 operating systems and even the VAX. Reasonable familiarity with the PDP-11 hardware and the RSX operating system will be assumed.

Speaker Biography:

Ed Cetron
Computer Systems Manager
Center for Biomedical Design
University of Utah
Salt Lake City, Utah

Ed Cetron is currently working on interfacing a PDP 11/44 to the Utah-MIT DEXTROS Robotic Hand. He began dealing with computers in a lab enviroment while studying for a bachlors degree at the University of Virginia. He was awarded a Masters Degree in Biomedical Engineering in 1983 for work on a "Microprocessor Control for Prosthetics."

Placing a Quote in an Indirect Symbol

Peter A. Shea
Systime. Ltd.

There are two ways to get a quote symbol into an indirect command file. Try, firstly:

```
@TI: "
AT.>.Enable Substitution
AT.>;'P1'
AT.>©Z
```

Or better still:

```
@TI:
AT.>.enable substitution
AT.>.setn Q 242
AT.>.sets QUOTE "'Q%V'"
AT.>;'QUOTE'
AT.>©z
```

Have fun, Pete Shea.

Placing a Quote in an Indirect Symbol

Simon Smith

Pauls Agriculture Ltd.
47 Key Street
Ipswich, IP4 1BX
Suffolk, England

Dear Sir:

After reading the short article by Roger Jenkins (RSX Multitasker May 1984 issue) concerning placing the quote character in an indirect symbol, may I be so bold as to suggest that the solution is simplicity itself. No messing around with user entry, or data files is required. I discovered the solution by accident. It went as follows.

"One sunny day, while creating a command file to perform yet another if thoise repetative jobs that always needs doing (hooray for ICP !!), the old fingers slipped while entering a .SETS line. I only discovered my mistake (or good fortune as it turned out to be) while looking back over the command file searching for another bug. I looked at what I had done in both disbelief and joy, for what I had entered, though technically incorrect, had not caused ICP to fall over with "Syntax Error" or the like!!"

The heart of the matter is that I had typed in ...

.SETS <symbol> #<string># and not .SETS <symbol> "<string>"

I thought, "I wonder. What would happen if I placed a quote in the string?". I did, and lo and behold, IT WORKED!!

The solution to this dilemma therefore, is to use the number sign (#) to delimit strings instead of the quote character.

Updated "Worst Bug I have Ever Encountered"

Mike Morrow
Clark Equipment Company
Automated Systems Division
P.O. Box 3000
Battle Creek, Michigan 49016

As the members of the RSX SIG may know, I won the prize for the "Worst Bug I have Ever Encountered" contest at the Fall '83 symposium in Las Vegas. In case you don't know, I found out the hard way that RSX11M+ checkpoints a common upon removal to its task image file, regardless of whether or not that file exists, resulting in bad things if that file or pack is not there any more.

Well folks, I think I have a bug (actually a nondesirable feature) to top even that one! We have a fairly large and complex program, written in MACRO using SUPERMAC, which consists of several modules that get assembled seperately and have the .OBJ files inserted into an .OLB, then task-built from that using overlays. Each module makes implied global references (.ENABL GBL) to subroutines and data elements in other modules and we let TKB pull the whole mess together. This arrangement in the past has worked satisfactorily.

When we rebuilt this task under RSX11M+ V2.1 and RMS V2.0 we experienced some strange problems. Two of these were isolated to RMS; DEC has had my SPR and PMD's for about three months now. However, one other problem had no apparent explanation, and it was in our application code.

While looking by chance at the listing of one of the modules in question, I saw an instruction

```
$CALL CSM
```

which was supposed to generate a call to a global subroutine named CSM in another module. Looking "just for grins" at the symbol table at the end of the listing I found no entry for CSM (I would have expected something like

CSM = ****** GX

to denote an implied global reference). I smelled a rat.

I got to a terminal and typed "MAC ,TI:=TI:", typed ".END" to shift MACRO to pass 2, then typed "JSR PC,CSM", the instruction that SUPERMAC translates "$CALL CSM" to. Instead of seeing

```
004767  000000G                 JSR PC,CSM
```

printed by MACRO, I was quite surprised to see

```
004767  007000                  JSR PC,CSM
```

I said to myself "Well that's nice. MACRO seems to think he knows just where my global subroutine will be when my program gets task built". I then realized that probably CSM is in MACRO's permanent symbol table, and was added to the new assembler that came with RSX11M+ V2.1. I got out the manual and went to the appendix that showed the PST, but found no CSM. I then went to my best processor handbook (PDP11 04/34a/44/60/70 1979-80) but found no CSM. I then DMP'ed MAC.TSK in R5 format and found the PST. Sure enough, there was CSM, along with TSTSET and WRTLCK, whatever they do. I then got an even newer handbook from a coworker, and it documented CSM as "Call to Supervisor Mode" for the 11/44.

My main gripe is that MACRO lets you use an opcode as an operand and doesn't tell you about it; it just assembles in the wrong code. This is compounded by the fact that MACRO keeps getting all these new CIS and FPS instructions added with new versions, tending to unpredictably break existing programs. I plan to submit a "suggested enhancement" SPR to DEC recommending that usage of an opcode as an operand be flagged with a warning; hopefully DEC will favorably respond.

I appreciate the opportunity to share this experience with the RSX users. I hope that other users put this in their bag of tricks in case they encounter this same situation themselves.

Serializing I/O in RSX-11M

Philip Miller and Robert Novas
Century Computing, Inc.
(301) 953-3330

Introduction

This article describes how I/O requests for two independent device controllers may be serialized. The need for serialization occurs in systems where simultaneous I/O through several high-speed controllers overloads the system bus, generating data late conditions.

A hypothetical example of a need for serialization is:

> A PDP-11 system has a high-speed disk, a high-speed magnetic tape, and a DR11-B DMA interface to an array processor. If all three devices run at once, the UNIBUS is saturated and data late errors are generated. Serializing I/O requests to the disk and the array processor would avoid the overload.

The technique described here involves no changes to user mode software, no changes to device data bases, and only minor changes to existing device drivers.

Restrictions

The technique presented here has several restrictions:

1. Only single controller drivers are supported. (However, multi-unit drivers are supported.)

2. Only two controllers may be serialized.

3. The two-character device names of the two controllers to be serialized must be different.

4. Every time a serialized driver is re-loaded, about 150 bytes of pool are lost.

5. Though minor, changes to existing device drivers are required. It is the authors' experience that many device drivers in the RSX world are delicately coded and difficult to change reliably.

Restrictions 1 though 4 may be eliminated by extending the code listed below, i.e., there are no conceptual barriers to doing a more general implementation.

Description

The basic concept is to imitate the manner in which the RSX $STMAP subroutine resolves contention for UNIBUS mapping registers. (RSX-11M-PLUS uses a similar technique to support mixed MASSBUS devices.)

Before a serialized driver initiates I/O, a call is made to REQCON (listed below) to request use of the controller. If the companion controller is idle, REQCON returns and the driver proceeds to initiate I/O; if the companion controller is busy, REQCON places the context of the driver in its "coordination block" and returns to RSX.

When a serialized driver receives control following completion of I/O, the driver releases the controller by calling RELCON (listed below). RELCON checks to see if the companion driver is waiting for the controller; if the driver is waiting, RELCON restores the saved context and continues execution of the companion driver.

Because the implementation here works for drivers that are loadable, code for RELCON must be below page 5 so that it can map between the drivers. Rather than add the module to the RSX monitor, we have opted to move the routine to the dynamic pool during driver initialization. To simplify the coding, we have placed RELCON in the coordination block, which is itself in the dynamic pool.

Data Structures

Each driver has a coordination block. The pointer to the block is placed at offset D.VPWF+2 in the driver dispatch table. (We do this in order to avoid changes to the standard RSX data base tables; users with user-written drivers can simplify the code here by placing the block pointer in the SCB.)

RELCON uses the "saved PC" in the other driver's coordination block to determine if the driver is waiting for the controller: a zero PC indicates that the driver is not waiting.

Installation Procedure

The following steps implement serialization of (hypothetical) devices D1 and D2:

1. In D1DRV.MAC, add the following lines directly after the fourth word of the $D1TBL (dispatch table):

```
CBPTR:: .WORD    0        ;Coordination block pointer
CORDEV:: .WORD   "D2      ;Device to coordinate with
```

2. In D1DRV.MAC, add "CALL REQCON" at a point in the logic with the following characteristics:

   - Before S.CTM is set (to start the device timeout).
   - Where it is known that a DMA data transfer is to be performed. (In general, it is not necessary or desirable to call REQCON for non-transfer operations.)
   - Where R0 may be destroyed.

3. Following every $IODON call in D1DRV, add "CALL RELCON". Note that RELCON destroys R0 and R1. (It is acceptable to call the RELCON subroutine even when you have not requested use of the controller.)

4. At the power fail entry point (D1PWF), add "CALL CBINIT". Note that the UC.PWF bit must be set in the UCB's U.CTL field to request that D1PWF be called upon driver initialization.

5. Task build D1DRV referencing the SERIAL object module (listed below). If the driver data base is contained in the D1DRV module, then SERIAL must be presented to the TKB before the D1DRV module.

6. Repeat steps 1-5 for D2DRV.MAC, using "D1" for the CORDEV value.

SERIAL.MAC

```
    .title  serial -- serialization subroutines
    .ident  /200/
    .enabl  lc
    .sbttl  sample driver coding
.if  eq 1                               ;comment block
;
;  Typical driver coding for a driver to coordinate
;  with the MT device.  The MT driver must have
;  similar coding to coordinate with this driver.
;
$drtbl:     .word   drini               ;standard RSX entry points
    .word   drcan
    .word   drout
    .word   drpwf
cbptr::     .word   0                   ;pointer to CBLK
cordev::.word       "MT                 ;name of coordinated device
    ...
;
;  Power fail driver initialization.
;
    call    cbinit              ;build coordination block

    ...
;
;  Request controller; for control functions that do not
;  require I/O, it is more efficient not to call this.
;  You must be at fork level to request controller.
;
;  A good way to find a place for this is to follow calls
;  to $STMAP (even if conditionalized on M$$EXT) with this
;  (unconditionalized) call.  $STMAP implies a transfer.
;
    call    reqcon                  ;request controller

    ...
;
;  Release controller; if this call is bypassed in the
;  I/O completion logic, the other driver will hang
;  forever.  Don't forget to make this call in timeout
;  and cancel logic.
;
;  The following logic should appear following every
;  $IODON/$IOALT call.  Note that it is okay to release
;  the controller even when you have not allocated it.
;
;  You must be at fork level to release controller.
;
    call    relcon                  ;release controller
.endc
```

SERIAL.MAC (continued)

```
        .sbttl  cbinit -- initialize CBLK
;
;  Allocate and build CBLK in dynamic memory.
;
;  Input:
;          R5=UCB pointer
;
;  Registers r3-r5 preserved.
;
cbinit::
        tst     cbptr                   ;already initialized?
        bne     50$                     ;brif yes
        mov     #c.bln,r1               ;r1=bytes to allocate
        call    $alocb                  ;allocate block
        bcs     100$                    ;brif no memory
        mov     r0,cbptr                ;set pointer to CBLK
        mov     r0,pact                 ;make pointer to ...
        add     #c.act,pact             ;activate routine
        mov     r0,pfind                ;make pointer to ...
        add     #c.find,pfind           ;find routine
        mov     #cblk,r1                ;source for move
        mov     #c.bln/2,r2             ;r2=words to move
30$:
        mov     (r1)+,(r0)+             ;move CBLK to pool
        sob     r2,30$                  ;loop
50$:
        return
;
;  No dynamic memory.   Not clear what to do here;
;  we mark the device off-line to prevent QIOs.
;
100$:
        bisb    #us.ofl,u.st2(r5)
        return
;
;
pact:   .blkw   1                       ;act pointer
pfind:  .blkw   1                       ;find pointer
```

SERIAL.MAC (continued)

```
        .sbttl  relcon -- release controller
;
;  Call here at fork level to release controller
;
;  Registers r2-r5 preserved.
;
relcon::
        mov     cbptr,r0                ;CBLK pointer
        clr     c.busy(r0)              ;no longer busy
        mov     cordev,r0               ;device to coordinate
        call    @pfind                  ;find other CBLK
        bcs     50$                     ;brif no other
        tst     c.pc(r0)                ;is other waiting?
        beq     50$                     ;brif not waiting
        call    @pact                   ;activate other driver
50$:
        return


        .sbttl  reqcon -- request controller
;
;  Registers r1-r5 preserved.
;  Call from fork level only.
;  Must call with (sp) equal to RSX return.
;
reqcon::
        mov     cordev,r0               ;name of other device
        call    @pfind                  ;find its CBLK
        bcs     50$                     ;brif no CBLK
        tst     c.busy(r0)              ;does other have cntrllr?
        beq     50$                     ;brif not
        mov     cbptr,r0                ;ptr to our own CBLK
        mov     (sp)+,c.pc(r0)          ;save callback
        add     #c.r1,r0                ;register save area
        mov     r1,(r0)+                ;save in our own CBLK
        mov     r2,(r0)+                ;
        mov     r3,(r0)+                ;
        mov     r4,(r0)+                ;
        mov     r5,(r0)                 ;
        return                          ;to caller's caller
50$:
        mov     cbptr,r0                ;pointer to our CBLK
        inc     c.busy(r0)              ;set that we have cntrllr
        return                          ;to caller
```

SERIAL.MAC (continued)

```
   .sbttl  coordination block
;
;  Coordination block.   Placed in dynamic memory so it
;  can map between drivers.  The block consists of a data
;  area and two subroutines  (find and act).
;
cblk:                                       ;start of CBLK
cpc:       .word   0                        ;callback pc (or zero)
busy:      .word   0                        ;0=free; 1=busy
map:       .blkw   1                        ;kisar5 of other driver
sr1:       .blkw   5                        ;r1-r5 of waiting driver
cptr:      .blkw   1                        ;pointer to other CBLK
;
;  offsets defining data in coordination block:
c.pc=cpc-cblk                               ;saved PC
c.busy=busy-cblk                    ;busy flag
c.r1=sr1-cblk                               ;register save area
;
;  Subroutine to find CBLK pointer of other driver.
;  Input:  r0=device name of other driver
;  Output: r0=CBLK pointer; C bit set if no CBLK
;  The code here must be PIC.
;
c.find=.-cblk                               ;offset in CBLK
find:
   mov     r1,-(sp)                 ;save
   mov     @#kisar5,-(sp)           ;save current mapping
   mov     @#$devhd,r1              ;r1=first DCB in chain
10$:       cmp     d.nam(r1),r0             ;is this the device?
   beq     50$                      ;brif yes
   mov     d.lnk(r1),r1             ;get next in chain
   bne     10$                      ;brif more
   br      70$                      ;error return
50$:       tst     d.dsp(r1)                ;driver resident?
   beq     70$                      ;brif not
   mov     d.pcb(r1),r0             ;r0=pcb address
   beq     60$                      ;brif inside RSX
   mov     p.rel(r0),@#kisar5       ;map to other driver
   mov     p.rel(r0),map            ;save locally for act
60$:       mov     d.dsp(r1),r0             ;r0=dispatch table
   mov     d.vpwf+2(r0),r0          ;r0=ptr to other CBLK
   beq     70$                      ;brif no CBLK
   clc                              ;success flag
   br      100$                     ;return
70$:
   sec                              ;error flag
100$:
   mov     (sp)+,@#kisar5           ;restore caller's map
   mov     (sp)+,r1                 ;restore
   return
```

SERIAL.MAC (continued)

```
    .sbttl  activate subroutine
;
;  Activate other driver.  (Resident in coordination block.)
;
;  Input:
;          r0=CBLK pointer of other driver
;
;  Registers r2-r5 preserved.
;
;  The code here must be PIC.
;
c.act=.-cblk                                    ;offset in block
act:
    mov     @#kisar5,-(sp)          ;save caller's mapping
    mov     map,@#kisar5            ;map to other driver
    mov     r2,-(sp)                ;save caller's r2-r5
    mov     r3,-(sp)                ;
    mov     r4,-(sp)                ;
    mov     r5,-(sp)                ;
    mov     r0,cptr                 ;save other CLBK ptr
    add     #c.r1,r0                ;r0=ptr to save area
    mov     (r0)+,r1                ;restore for other
    mov     (r0)+,r2                ;
    mov     (r0)+,r3                ;
    mov     (r0)+,r4                ;
    mov     (r0),r5                 ;
    mov     cptr,r0                 ;restore for...
    inc     c.busy(r0)              ;set busy
    call    @c.pc(r0)               ;call other driver
    mov     cptr,r0                 ;r0=ptr to other CBLK
    clr     c.pc(r0)                ;no longer waiting
    mov     (sp)+,r5                ;restore for caller
    mov     (sp)+,r4                ;
    mov     (sp)+,r3                ;
    mov     (sp)+,r2                ;
    mov     (sp)+,@#kisar5          ;restore caller's map
    return                          ;to releasing driver
;
c.bln=.-cblk                                    ;size of CBLK
    .end
```

USING FMS-11 FROM RSX-11M-PLUS INDIRECT COMMAND FILES

Mark Chatterton
General Mills, Inc.
9000 Plymouth Ave. N.
Minneapolis, MN 55427

INTRODUCTION

At General Mills, we have found that Digital's FMS-11 software provides a very good user interface for programs that require keyboard input. The ability to move forwards and backwards to various fields on the screen, along with definable HELP screens and messages make FMS an invaluable tool for our applications.

When I read in one of Allen Watson's columns that the Indirect Command Processor for RSX-11M-PLUS contained a Form Driver interface, I was intrigued. Unfortunately, documentation for this handy feature is mysteriously lacking in the RSX books. The Indirect command file manual does little more than mention the fact that a Form Driver interface exists.

My goal here is to provide enough information so that a reader familiar with FMS software can access the Forms Driver from within an Indirect command file. This information was discovered, for the most part, through trial and error.

FORMS & FORMS LIBRARIES

Forms to be accessed through Indirect are created by the Form Editor (FED) and maintained with the Form Utility (FUT) just as they would be for any other programming language.

The default Indirect provides a 4000 byte data area for FMS use. If your screen exceeds this limit, unpredictable results occur.

COMMAND SYNTAX

Indirect uses the ".FORM" command to access the Form Driver. The syntax of the command is:

```
.FORM FNC,PARAM1,PARAM2,...,PARAMN
```

Where:

FNC is a three letter code telling Indirect which Form Driver function you want to perform.

PARAM1 through PARAMN are parameters to be passed to to the Form Driver so it can execute the function.

Every parameter has a required type: string or numeric. String variables or quoted strings may be used for string parameters. Numeric variables or numbers may be used for numeric parameters. String substitution may be used.

ERROR RETURNS

After each ".FORM" command is executed, Indirect sets two of its special numeric symbols to indicate the success or failure of the operation. The symbol <FILERR> will be greater than zero if the command completed to success. If <FILERR> is less than zero, some error occured, and a list of the error codes are in the FMS manual. If <FILERR> is equal to -4. or -18., an error occured opening or reading your form library. The FCS or RMS error code will be placed in a symbol called <FILER2>.

FUNCTION CODES & PARAMETERS

What follows is a list of all the Form Driver function codes that I know about and the parameter list associated with each code. Note that the function mnemonics are identical to the ones used by the Macro-11/FMS interface. I've indicated optional parameters with square brackets. If you choose to omit an optional parameter, be sure to include the comma that immediately follows it. Also remember to use the correct symbol type, string or numeric, for each parameter you include.

A complete discussion of what each function does can be found in the FMS manual.

For an example of all this, look at the module "FMSDEM" in LB:[1,2]INDSYS.CLB of your distribution kit.

ALL - Get the responses for all fields

.FORM ALL,[fval],[term]

Inputs:

fval (string) = The concatenated values for all fields in the form. If null, the values are

only stored in the Form Driver data area. You can then access them later using RTN or RAL.

Outputs:
term (numeric)= Terminator code for the key that the operator used to end input.

ANY - Get the response for any field

.FORM ANY,[fid],[fidx],[fval],[term]

Inputs: None

Outputs:
fid (string) = Name of the field filled by operator.
fidx (numeric) = Field index of the field filled by operator (if that field is indexed).
fval (string) = The value entered by the operator.
term (numeric) = Terminator code for the key that the operator used to end the input.

CLS - Close a form library

.FORM CLS

CSH - Clear entire screen and show a form

.FORM CSH,fnam,[line]

Inputs:
fnam (string) = Name of the form.
line (numeric) = Starting line for the form, overriding the line number assigned with FED.

Outputs: None

DAT - Get named data

.FORM DAT,[fid],[fidx],[fval]

```
    Inputs:
        fid (string)   = The name of the data to be retrieved.
        fidx (numeric) = The index value of the data to be
                         retrieved.
        (At least one of these parameters must be present. If
         you include both, fidx will be ignored)

    Outputs:
        fval (string) = The named data value retrieved from
                        the form.
```

GET - Get the response for a specified field

```
    .FORM GET,fid1,[fidx1],[fid2],[fidx2],[fval],[term]

    Inputs:
        fid1 (string)   = The field name.
        fidx1 (numeric) = The field index for fid1 (when the
                          field is indexed).

    Outputs:
        fid2 (string)   = The field name.
        fidx2 (numeric) = The field index for the fid2 (when
                          the field is indexed).
        fval (string)   = The field value.
        term (numeric)  = Terminator code that the operator
                          used to end the input.
```

GSC - Get current line of scrolled area

```
    .FORM GSC,fid,fval,[term]

    Inputs:
        fid (string) = Name of the field within the scrolled
                       area.

    Outputs:
        fval (string)  = The concatented values for all fields
                         in the line.
        term (numeric) = Terminator code that the operator
                         used to end the input.
```

LST - Output to the bottom line of the screen

```
.FORM LST,[fval]

Inputs:
    fval (string) = String to be displayed on the bottom
                    line. If null, bottom line will be
                    cleared.

Outputs: None
```

OPN - Open a form library

```
.FORM OPN,flnm

Inputs:
    flnm (string) = A form library file specification.

Outputs: None
```

PAL - Output data to all fields

```
.FORM PAL,[fval]

Inputs:
    fval (string) = The concatenated values to be displayed.
                    If null, the Form Driver restores de-
                    fault values for all fields.

Outputs: None
```

PSC - Output data to current line of scrolled area

```
.FORM PSC,fid,[fval]

Inputs:
    fid (string) = Name of the field within the scrolled
                   area.
    fval(string) = The field values to be displayed. If
                   null, the Form Driver restores default
                   values to the field.

Outputs: None
```

PUT - Output data to a specific field

.FORM PUT,fid,[fidx],[fval]

Inputs:
fid (string) = The field name.
fidx (numeric) = The field index for the field (when the field is indexed).
fval (string) = The field value to be diplayed. If null, the Form Driver restores the default value to the field.

Outputs: None

RAL - Return the responses for all fields

.FORM RAL,fval

Inputs: None

Outputs:
fval (string) = The concatenated vaues for all fields in the form.

RTN - Return the response for a specific field

.FORM RTN,fid,[fidx],fval

Inputs:
fid (string) = The field name.
fidx (numeric) = The field index for the field (when the field is indexed).

Outputs:
fval (string) = The field value.

SHO - Show a form

.FORM SHO,fnam,[line]

Inputs:
fnam (string) = A form name.
line (numeric) = Starting line for the form, over-

riding the line number assigned with FED.

Outputs: None

SPF - Turn supervisor-only mode off

.FORM SPF

SPN - Turn supervisor-only mode on

.FORM SPN

TRM - Process field terminator

.FORM TRM,[fid1],[fval],term,[fid2],[line]

Inputs:

fid1 (string)	=	The field name. Required if a scrolled area terminator is specified.
fval (string)	=	Data to be displayed in the top or bottom line of the scrolled area. Ignored if specified terminator is not a scrolled area terminator.
term (numeric)	=	Numeric code for the terminator to be processed.

Outputs:

fid2 (string)	=	Name of the new current field.
fidx (numeric)	=	Index of the new current field.

FINAL COMMENT

I have not yet been able to duplicate my favorite FMS function through Indirect. The FORTRAN Form Driver interface allows the use of the GET function (FGET in FORTRAN) without parameters. When this call occurs, the Form Driver places the cursor in the lower-right corner of the screen and waits for the operator to hit the ENTER or RETURN key. This is a great way to synchronize the task with the operator.

Unfortunately, the Indirect version of the Form Driver inter-

face gets very upset when you don't pass it a field name with the GET call. My way around this is to define a single character field in the lower-right corner of the screen, call GET using this field, and wait for the operator to hit ENTER.

Good-Luck !!

Spiffy Things to Do with Virtual Disks

James A. McGlinchey
Software Engineering Consultant
Post Office Box 451
Warrington, PA 18976
(215) 348-7261

Virtual Disks have been around in the RSX world for a while now, yet practical uses for them seem to evade a lot of users' imaginations. I want to present some techniques I have used in the past two years which have added to the flexibility and reliability of systems I support.

What's a Virtual Disk?

Simply stated, a Virtual Disk is a contiguous file on an RSX disk that is in itself treated as a disk. A Virtual Disk is driven through a Virtual Disk driver, which remaps all Virtual Disk accesses into logical disk accesses and re-queues the request to the physical disk driver in a method entirely transparent to the user.

A Virtual Disk can have BAD run against it (if fact, it MUST), can be INItialized, have UFD's placed on it, and can be backed up and copied using BRU. The only thing you can't do with a Virtual Disk that you can do with a physical disk is FORMAT it!

The principle of Virtual Disks was first demonstrated by Ralph Stamerjohn; he was the first to release a Virtual Disk package. I like the package placed on the Fall 1982 (Disneyland) RSX SIG Tape by Glen Everhart of RCA, for a couple of reasons: (1) It runs on both RSX-11M and RSX-11M-PLUS, and (2) I have been using it for a year without trouble.

Virtual Disk packages are fairly easy to use. Although it certainly helps to understand how they work, it really is not necessary in order to use them. The best way to come to an understanding of Virtual Disks is to bring up a package and tinker

with it a bit.

I don't want to dwell on the theory of Virtual Disks; I want to get on to some practical uses.

Practical use No. 1: A Poor Man's Disk Quota System

Got users who leave their old listings lying around on your disk, and then complain to you when they run out of space? Give them each a Virtual Disk to use, and then the problem becomes theirs rather than yours (that's the whole secret of being a successful System Manager, isn't it?). A small change to the HEL task will enable you to assign them a Virtual Disk when they log on.

Practical use No. 2: Partitioning a Big Disk

Create Virtual Disks for each major application or Data Base. Keep all the files associated with the application on the same Virtual Disk. They will thus be isolated from the rest of the files on the physical disk, but will still be easily accessible. A Virtual Disk gets a complete set of UIC's, so you can use lots of UFD's to arrange your software in an orderly manner.

Practical use No. 3: Protecting and Separating Distribution Kits

In a Big Disk system, it is often the case that the distribution kits for layered products are left on the system disk. After all, what's all that space for, if it isn't to be filled up? You then wind up with the distribution kits for Datatrieve, FMS, and FORTRAN-77 scattered all over your system disk, and oftentimes it isn't obvious which UIC goes with which layered product. If you give each layered product its own Virtual Disk to live on, each kit is kept logically separate from the others.

Practical use No. 4: Keeping RSX-11S by itself

RSX-11S makes life sticky because it's an operating system, not just a layered product. It can share UIC's with its host operating system, and its distribution kit can therefore get all intertwined with the files which belong to the host. Murphy's Law says that when you try to move an 11S system off the host system's disk, you'll probably get part of the host system along with it, and you'll leave part of the 11S system behind. Put you" RSX-11S Distribution kit on its own Virtual Disk, and then put each of the generated target systems on their own Virtual Disks. By the way, put the target applications on the same Virtual Disk as its target Operating System, and you'll avoid the mismatch problem that often occurs when you have to support multiple target systems from a single host.

Practical Use No. 5: Disk-to-Disk Backup

Got a Big Disk system, and your users really get steamed when you have to take the system down to do backups? Try creating a Virtual Disk and use a command file to do a disk-to-Virtual Disk incremental backup during the night, then come morning you can trundle in and copy the Virtual Disk to tape at your (and your users') leisure.

Practical Use No. 6: Shadow Recording

Shadow Recording on M-PLUS has a couple hidden costs. At first it sounds really super, almost like getting it for free. But wait - what if you only have one disk? Sorry, says DEC, gotta get a second one, and it must be exactly the same type as the first one. That may mean getting a second RA80, at a cost of $OUCH. ! To add

insult to injury, you must shadow the entire disk, even if only a few files are really critical to you.

Virtual Disks can help here. You can have two different physical disks, but you can shadow one Virtual Disk to another Virtual Disk, independent of the physical type of disk the Virtual Disks live on (got that?). For instance, suppose you have an RL02 and an RM02, and you want to shadow record about 8 MB worth of files. Set up a Virtual Disk on the RM02 and copy the live files into this Virtual Disk. Then set up a Virtual Disk on the RL02 that is exactly the same size as the Virtual Disk on the RM02. Then Designate the second Virtual Disk as the Shadow of the first Virtual Disk.

I tried this, and it works, with one GOTCHA. You have to know a little bit about device drivers here. You have to go into the Virtual Disk package and build the two Virtual Disk units you want to use for shadowing with their own separate Status Control Blocks (SCBs). Other than that, it works just fine.

Practical Use No. 7: Multiple SYSGENs

Support a lot of systems? Tired of traipsing all over the plant to do the SYSGENs? Take heart. Also take a Big Disk and create on it a lot of Virtual Disks which are the size of the physical disks on your target systems. Put your RSX Distribution kit on another Virtual Disk, just to keep it separate. Then you can set up your SYSGENs using the Saved Answer Files, and turn a whole bunch of them loose overnight. Come morning, all you have to do is to DSC or BRU the Virtual Disks off onto tapes, boogie on out to the target system, copy the tape onto the real physical disk, BOO the new system and SAV it.

DECUS

DECUS SUBSCRIPTION SERVICE
DIGITAL EQUIPMENT COMPUTER SOCIETY
249 NORTHBORO ROAD, (BPO2)
MARLBORO, MA 01752

Bulk Rate
U.S. Postage
PAID
Permit No. 18
Leominster, MA
01453

MOVING OR REPLACING A DELEGATE?

Please notify us immediately to guarantee continuing receipt of DECUS literature. Allow up to six weeks for change to take effect.

( ) Change of Address
( ) Delegate Replacement

DECUS Membership No.: _____
Name: _____
Company: _____
Address: _____
_____
State/Country: _____
Zip/Postal Code: _____
Phone No.: _____

Mail to: **DECUS - Attn: Subscription Service**
249 Northboro Road, (BPO2)
Marlboro, MA 01752 USA