RSX

MULTI-TASKER

MAY 1985 ISSUE

RSX MULTITASKER

Table of Contents

On-Line Pool Analyzer Patch

Scott Snadow, chair RSX OCLUG
General Dynamics
P.O. Box 2507
Mail Zone 4-68
Pomona, CA 91769
(714) 620-7511 X4779

I have a "little patch" to a popular RSX-11M DECUS program, OPA (the On-line Pool Analyzer); I believe that the 11M version hasn't been updated since the V3.2 days (although the M+ version is flourishing.)

As a base, start with the Spring '83 RSX SIG Tape, UIC [370,210]. There, you will find a submission by Rick Webster, which uses SLP to apply patches to J. Neeland's original OPA submission on an earlier tape (a "virgin" copy of Neeland's OPA is in the same account as Webster's patches, so you DON'T need to track down the earlier tape; Webster names the virgin copy OPA.VGN).

I found only one difference between V4.0 and V4.1, as far as OPA is concerned: the VCB (Volume Control Block) grew slightly in V4.1. Because the change to OPA was so small, I didn't want to create another submission ("Snadow's OPA"?); but here's the changes that I made:

I edited OPA.VGN:

1) Find the line

```
                MOV     #VCBSTG,R1      ;MARK WITH '<VCB -
```

   and change it to:

```
                MOV     #VCBSTG,R1      ;MARK WITH '<VCB   -
```

   (NOTE: I'm only changing the comment, adding extra spaces)

2) Find the line (near the end)

```
        VCBSTG: .ASCIZ  /<VCB  - /
```

   and change it to:

```
        VCBSTG: .ASCIZ  /<VCB   -  /
```

   (NOTE: I'm just adding one extra space on each side of the "-")

The other change that is necessary (I don't recall if it appeared in 4.0 or 4.1) is that devices can now be mounted /FOREIGN, just like in M+. If a device IS mounted /FOREIGN, OPA will usually crash your system. The fix for this is also small:

1) Find the section of code

```
10$:    BIT     #DV.MNT,U.CW1(R0) ;CHECK IF DEVICE IS LEGAL FILES-11
   BEQ     NOWB            ;NOT IF NOT MOUNTABLE
   BIT     #DV.F11,U.CW1(R0)
   BEQ     NOWB            ;NOT IF NOT FILES-11
   BITB    #US.MNT,U.STS(R0) ;BUT IS IT ACTUALLY MOUNTED?
   BNE     NOWB            ;NO IF BIT ON!
```

and add the following two lines, immediately following the above code:

```
   bitb    #us.for,u.sts(r0) ;But is it mounted foreign?
   bne     nowb            ;Yes if bit on!
```

2) Find the section of code

```
TSTUCB: BIT     #DV.MNT,U.CW1(R2) ;IS THIS DEVICE MOUNTABLE?
   BNE     10$             ;Yes - branch
   JMP     NXTDEV          ;IF NOT, GO TRY FOR ANOTHER DEVICE
10$:    BIT     #DV.F11,U.CW1(R2) ;IS IT FILES-11?
   BNE     20$             ;Yes, branch
   JMP     NXTDEV          ;IF NOT, GO TRY FOR ANOTHER DEVICE
20$:    BITB    #US.MNT,U.STS(R2) ;OKAY, IS IT ACTUALLY MOUNTED
   BEQ     30$             ;Yes, branch
   JMP     NXTUNT          ;IF NO, CHECK THE NEXT UNIT
30$:    MOV     U.VCB(R2),R0    ;AHA!, GOT ONE.  GET ADDRESS OF VCB
```

Change the label on the last line (30$:) to 40$, and insert the following 3 lines just before the NEW line 40$:

```
30$:    bitb    #us.for,u.sts(r2) ;But is it mounted foreign?
   beq     40$             ;Branch if no.
   jmp     nxtunt          ;If foreign, check next unit.
```

Then, go ahead and run OPABLD.CMD.

The RSX System Manager

Recovering From Disk Disasters, Part I

Let's face it. At one time or another, we have all deleted files that we wish we hadn't. A good number of requests I get from users of RSX deal with recovering deleted files. This month, I will discuss what happens when a Files-11 file is deleted and describe two methods which have been implemented to recover deleted files.

1.0 HOW THE FILES-11 ACP DELETES FILES

When PIP (or DCL DELETE) is called to delete a file, or when a file is deleted from a program (FCS DELET$ or RMS $ERASE calls), two operations occur. First, the directory entry for the file in the User File Directory (UFD) is located and then removed. Next, the Files-11 ACP is called with the IO.DEL function to actually mark the file as deleted and release the allocated file space. In detail, the IO.DEL operation performs the following operations:

1. Validate the file ID passed in the IO.DEL QIO.

2. Release the blocks allocated by the file. The retrieval pointers in the file header are scanned, and the bits in the bitmap file corresponding to the logical blocks used by the file are marked free.

3. Invalidate the file header. This is done by clearing the words containing the file ID and the file header checksum.

4. Update the file header bitmap to make the file header available.

The mechanism used by the ACP to delete files actually makes file recovery straightforward. After all, pertinent file information is still intact in the header, including identification, file ownership and protection, record attributes, and the mapping to the logical blocks used by the file. Additionally, deleted file headers are easy to recognize by the zeros stored in the file ID and checksum words. The trick then, is to "undo" or ignore what the ACP has done.

2.0 RECOVERING DELETED FILES

Assuming a desired file has been deleted, and there is no satisfactory backup copy, a user has to act quickly in order to recover the file. This is because the file header and the blocks used by the file are available for use by other tasks when creating and extending files. In fact, once the file header has been reused, there is little the user can do, except for performing a block by block scan of the volume [1].

To my knowledge, at least three programs have been written to recover deleted files; two have been published in the Multi-Tasker [2, 3]. Unfortunately, I do not have access to a copy of the December, 1980 issue containing P. G. Hansell's UNDELETE program.

Note that in the following descriptions, when a UIC is specified for a deleted file, it refers to the owner of the deleted file, which is not necessarily the same as the directory which previously contained the file. The file owner UIC is determined when the file is created, or when a PIP operation using the "/FO" switch is made.

2.1 Method 1: "In-Place" UNDELETE Program

One way to recover deleted files is by using an "in-place" recovery method. This means the deleted file is resurrected on the volume which it resides, preserving the original file header and the allocated blocks. This program, called UNDELETE and written by Larry Baker, works as follows:

1. Open the volume's index file for write access (the volume must be mounted with the /UNL switch), and read the home block to get the location and size parameters for the file header section of the index file.

2. Obtain a file specification from the user, including UIC, filename, extension, and version. By default, all deleted files on the volume are recovered. The syntax is similar to SRD's filename selection criteria, where "?" matches any single character, and "*" matches any number of characters.

3. Proceed sequentially through the index file, looking for deleted file headers (zeros in the file ID and checksum words). For every deleted file header, see if the file qualifies under the selection criteria.

4. If the file qualifies, re-validate the file header by restoring the file ID word in the header and recomputing the checksum. Rewrite the file header and mark the header as being used in the file header bitmap. Go back to the previous step.

When UNDELETE exits, the recovery job is only partially completed. The blocks used by the file are still marked as being free, and the file is "lost," since it is not contained in any directory. The next step is to run VFY, first to rebuild the storage allocation bitmap, and second to enter the lost file in UFD [1,3].

When the VFY /UPDATE pass is run, the recovered files will show up as having all of their blocks marked free. This is the normal behaviour. Problems can occur if blocks from the recovered files have already been allocated by other files. This is where things can get pretty ugly. VFY will start printing messages about multiply allocated blocks, and will automatically perform an additional pass over the volume, doubling the time to execute VFY. Furthermore, the only remedy for multiply allocated blocks is to manually delete one of the files that shares blocks.

After updating the storage allocation bitmap, a VFY lost file scan is executed. All recovered files (along with any other lost files) will be entered in the [1,3] UFD. Finally, it is best advised to run another VFY verification pass to make sure the index file and the bitmap file agree on the number of blocks that are either free or allocated.

Although we have used UNDELETE several times to save our necks, the program does have its disadvantages:

1. No users or tasks can create or write files on the volume until the entire operation completes, including the VFY phase.

2. Performing a "wild-card" UNDELETE can be a nightmare, since files that were deleted long before can be resurrected, which will cause tremendous problems with the file block allocation checks when VFY is run.

3. The operation is time-consuming. Anyone who has large disk drives knows how long it takes VFY to perform a verification scan on a large volume (sometimes 45 minutes on an RM05).

4. UNDELETE cannot handle multi-header files. I do not know what would happen if a multi-header file was processed using UNDELETE. I imagine VFY would try to process the first extension header, would find it deleted, and end up making a mess of everything.

2.2 Method 2: Kirkman's UND Utility

The second method of file recovery is very straightforward. Since the information in the file header is nearly complete, a program can process the logical block map of the deleted file, read the deleted blocks (using logical block I/O), and write them to another volume. The file attributes of the deleted file can also be read and stored with the new file.

The detailed flow of UND, written by Richard Kirkman, is as follows:

1. Obtain a command line of the form

   outdev: = indev: [ggg,mmm]

   Where "outdev:" and "indev:" are different disk volumes, and "[ggg,mmm]" specifies the UIC of the owner of the deleted file(s). Note that all deleted files owned by this UIC will be recovered, so plenty of free space on the output volume may be required.

2. Open the index and bitmap files, and point to the first header in the index file.

3. Read a file header. If the file header is deleted (zeros in the file ID and checksum words), and the file owner matches the UIC on the command line, then open a new file on the output volume and copy the contents of the deleted file to the output file. UND checks the bitmap file to see if any of the contents of the file have been reallocated and possibly reused. If so, a warning message is displayed.

4. If there are more file headers, go back to the previous step; otherwise, exit.

UND is a powerful little utility. It is even helpful as a Files-11 learning aid to show how the index file is organized and how the logical blocks on a volume are organized into coherent files. However, there are a few minor problems:

1. There is no support for recovery of multi-header files. Only the first file header of a file is processed. Therefore, if a large file is recovered using UND, the user must verify that all of the data was transferred.

2. All deleted files on the volume corresponding to the input UIC will be recovered. This is not a serious problem for small (e.g. RL02) volumes. However, I can imagine what might happen if UND was used on an RP06 or RP07 with a directory that has a high level of file creations and

deletions.

All of these complaints are minor, of course. The following section will address the problems and provide some solutions.

The most recent version of UND that I am aware of can be found on the Spring 1983 RSX SIG tape in UIC [370,210]. This version was originally written by Richard Kirkman, modified by John Hayes, and further modified and submitted by Rick Webster of Caterpillar Tractor Company.

3.0 MODIFICATIONS TO KIRKMAN'S UND

I am presently in the process of modifying UND to provide more functionality and to eliminate some of its annoying features. Although I cannot make any commitments at this time, I intend to submit these modifications to the Spring 1984 SIG tape in UIC [307,20].

3.1 Support For Multi-Header Files

Adding support for multi-header files is relatively straightforward, especially when the source files for the Files-11 ACP are available. After copying the blocks mapped by the first file header, UND checks to see if an extension header exists; if so, the current index file position is saved, all of the file's extension headers are processed, and the index file position is restored.

The only serious error that can occur while recovering a multi-header file is if one of the extension headers has been reused. If this happens, UND will print an error message and abort the file recovery.

3.2 Better UIC And Filename Selection

Support for the "*" wildcard in the UIC specification, along with "*" and "?" wildcards in the filename and file type fields will provide more control over selection of deleted files. Additionally, an explicit or "*" version number can be supplied.

3.3 More Switches

Even though "we want to stamp out switches in your lifetime," UND could use at least three.

"/HELP" is self-explanatory (even though DEC utilities have never heard of it).

"/LIST" prints out the filenames of all files that qualify under the selection criteria, without actually recovering the files. This would be useful if there is a limited amount of free space on the output volume.

"/CONFIRM" causes a prompt to be issued to the terminal when a file which is a candidate for recovery is found. The standard responses, "Y" for Yes, "N" for No, "Q" for Quit, and "G" for Go are included.

4.0 SUMMARY

Tools for recovering deleted files should be required only when a few recently created files are mistakenly deleted. This typically happens when a well-meaning user types a command of the form "PIP *.*;*/DE" and immediately notices the error.

Frequent reliance on these tools indicates a problem in system management: insufficient user training, infrequent backup procedures, or plain carelessness.

In future articles, we will explore some of these issues, and tackle some more frightening examples of disk disasters.

****

Next month: Customizing printer flag pages and other enhancements to the queue manager.

****

References:

[1] Doran, Chris, "Last-Ditch Method for File Recovery," Multi-Tasker, Vol. 15, No. 7, February 1982, pp. 41-43.

[2] Kirkman, Richard, "UND, a Program to Undelete Files," Multi-Tasker, Vol. 15, No. 1, July 1981, pp. 12-18.

[3] Hansell, P. G., "UNDELETE Program for RSX Disks," Multi-Tasker, Vol. 14, No. 1, December 1980.

[4] Digital Equipment Corporation, "Files-11 On-Disk Structure Specification," Multi-Tasker, Vol. 15, No. 9, April 1982, pp. 42-76.

****

Please send questions, comments, ideas and submissions for this column to the following address:

Gary Maxwell
U.S.G.S. M/S 977
345 Middlefield Road
Menlo Park, CA 94025

The Bag of Tricks: Macro-11

Bruce R. Mitchell
Machine Intelligence and Industrial Magic
PO Box 601
Hudson, WI 54016

This column covers MACRO-11 bag-of-tricks routines, as stated in previous issues of the Multi-Tasker. All MACRO programmers are encouraged to submit their favorite routines to the Multi-Tasker so that these useful, interesting, or just plain bizarre tricks can be put out before the SIG in general for the admiration and edification of all.

In this month's column, we have something which none of us really need, since we all write perfect code - a routine to do prettified dumps when the host program loses it entirely.

When a MACRO program loses it and does an IOT, odd address trap, or whatever, we get back a generally interesting but not very readable message from the terminal which dumps all the registers at the time the fatal error occurred. This is neat for those who understand what it's telling them, but not so neat for the poor sucker out in the field who doesn't know an APR from his mother.

This routine gives dumps of ALL the information available when a task goes bonkers. In particular, the walkbacks from memory protect violations are a beautiful thing to see - there is much useful information there.

It should be noted that this routine makes memory references, in some cases, PSECT-base relative through use of the variable SVTK. Normally SVTK should be equated to the entry point of the program so that any errors occurring in the main are made relative to the base of the code area. This eliminates the need for a map when such an error occurs, since the address of the instruction is then the same as it was on the assembly listing!

It should also be noted that a DIR$ ICONCK must be issued in the mother program - if this is not done, the trap catching routines will never be called!

This routine was written from scratch, and there is no claim to it being beautiful or even very nice. It dumps all registers even if some of them are not valid for the particular trap which occurred. However, it suits most uses, since the responsible programmer will know if the extended registers are valid or not from the message printed before the dump began.

Because the author tends to separate data and code structures in his programs, the labels have been made so that the data and code can be readily separated within a single source file.

This is an interesting "subroutine", since it never returns. It dies a gory death, assuming that the error was fatal, which is probably true in most cases. Bold programmers could perhaps use the saved SR0, SR1 ... etc to attempt error recovery.

```
;  Macro to load a CO: print QIO DPB and execute the DPB

   .MACRO  CPRINT  STRING                    ; Begin CPRINT macro

   MOV     #STRING, CONOUT+Q.IOPL
   MOV     #STRING'L, CONOUT+Q.IOPL+2
   DIR$    #CONOUT

   .ENDM                                     ; End CPRINT macro


   .PAGE
   .SBTTL  Directive Parameter Blocks


;  Exit to RSX with error status

EXERR:     EXST$   EX$ERR
```

```
;  Marktime DPBs

MARKTM:     MRKT$   MKEFN, 5, 2

;  QIO DPBs

CONOUT:     QIOW$   IO.WVB, COLUN, COEFN,, IOSTAT,, <0, 0, 40>

;  Specify SST vector table DPBs

ICONCK:     SVTK$   ICNTBL, ICTBLN

;  Wait for event flag DPBs

WAITFR:     WTSE$   MKEFN


    .PAGE
    .SBTTL  Messages and Strings

;  Messages for invoking terminal or system console

S24:        .ASCII  \SVTK-F-SSF, Specify SST vector table (SVTK$) failed\
S24L = . - S24

S35:        .ASCII  \SVTK-F-SC0, SST error - Odd address or nonexistent memory\
S35L = . - S35

S36:        .ASCII  \SVTK-F-SC1, SST error - Memory protect violation\
S36L = . - S36

S37:        .ASCII  \SVTK-F-SC2, SST error - T-bit trap or breakpoint (BPT)\
S37L = . - S37

S38:        .ASCII  \SVTK-F-SC3, SST error - I/O trap (IOT)\
S38L = . - S38

S39:        .ASCII  \SVTK-F-SC4, SST error - Reserved instruction\
S39L = . - S39

S40:        .ASCII  \SVTK-F-SC5, SST error - Non-RSX emulator trap (EMT)\
S40L = . - S40

S41:        .ASCII  \SVTK-F-SC6, SST error - TRAP instruction\
S41L = . - S41

S42:        .ASCII  \SVTK-F-SC7, SST error - Floating point exception\
S42L = . - S42
```

```
S43:        .ASCII  \           Program counter (PC):  \
S43A:       .BLKB   6
S43L = . - S43

S44:        .ASCII  \           Processor status word (PSW):  \
S44A:       .BLKB   6
S44L = . - S44

S45:        .ASCII  \           Memory protect status register (SR0):  \
S45A:       .BLKB   6
S45L = . - S45

S46:        .ASCII  \           Virtual PC of faulting instruction (SR2):  \
S46A:       .BLKB   6
S46L = . - S46

S47:        .ASCII  \           Instruction backup register (SR1):  \
S47A:       .BLKB   6
S47L = . - S47

S48:        .ASCII  \           Instruction operand lower byte:  \
S48A:       .BLKB   6
S48L = . - S48

S49:        .ASCII  \           Program counter, SVCODE PSECT relative:  \
S49A:       .BLKB   6
S49L = . - S49

S50:        .ASCII  \           Virtual PC (SR2), SVCODE PSECT relative:  \
S50A:       .BLKB   6
S50L = . - S50

S51:        .ASCII  \           Directive status word ($DSW):  \
S51A:       .BLKB   6
S51L = . - S51

S52:        .ASCII  \           Processor general register 0 (R0):  \
S52A:       .BLKB   6
S52L = . - S52

S53:        .ASCII  \           Processor general register 1 (R1):  \
S53A:       .BLKB   6
S53L = . - S53

S54:        .ASCII  \           Processor general register 2 (R2):  \
S54A:       .BLKB   6
S54L = . - S54

S55:        .ASCII  \           Processor general register 3 (R3):  \
S55A:       .BLKB   6
S55L = . - S55

S56:        .ASCII  \           Processor general register 4 (R4):  \
```

```
S56A:       .BLKB   6
S56L = . - S56

S57:        .ASCII  \           Processor general register 5 (R5):  \
S57A:       .BLKB   6
S57L = . - S57
   .EVEN


   .PAGE
   .SBTTL  Tables


;  SST Internal Consistency Error Vector Table
;
;  Form:  Service routine vectored on SST

ICNTBL:     .WORD   SST000                  ; Odd address trap
   .WORD    SST001                  ; Memory protect violation
   .WORD    SST002                  ; T-bit or BPT
   .WORD    SST003                  ; IOT
   .WORD    SST004                  ; Reserved instruction
   .WORD    SST005                  ; Non-RSX EMT
   .WORD    SST006                  ; TRAP
   .WORD    SST007                  ; Floating point exception
ICTBLN = . - ICNTBL


   .PAGE
   .SBTTL  Individual Variables


SDSW:       .WORD   0                       ; SST storage word for $DSW
SIOP:       .WORD   0                       ; SST storage word
SR0:        .WORD   0                       ; SST storage word for R0
SR1:        .WORD   0                       ; SST storage word for R1
SR2:        .WORD   0                       ; SST storage word for R2
SSR0:       .WORD   0                       ; SST storage word
SSR1:       .WORD   0                       ; SST storage word
SSR2:       .WORD   0                       ; SST storage word


   .PAGE
   .SBTTL  SST00x  Internal SST Error Handling


;   SSSSSSSSS   SSSSSSSSS  TTTTTTTTTT   0000000 0   0000000 0
;   SSSSSSSSS   SSSSSSSSS  TTTTTTTTTT   00000000    00000000
;  SS          SS             TT      00     000  00     000
;  SS          SS             TT      00    0 00  00    0 00
;   SSSSSSSS    SSSSSSSS      TT      00   0  00  00   0  00  xx      xx
;   SSSSSSSS    SSSSSSSS      TT      00   0  00  00   0  00   xx    xx
;          SS          SS     TT      00  0   00  00  0   00    xx  xx
```

```
;          SS          SS      TT      00  0   00  00  0   00     xxxx
;          SS          SS      TT      00 0    00  00 0    00     xxxx
;          SS          SS      TT      000     00  000     00    xx  xx
;   SSSSSSSS    SSSSSSSS       TT       00000000    00000000    xx    xx
;   SSSSSSSS    SSSSSSSS       TT      0 0000000   0 0000000   xx      xx
```

```
;  SST00x - Internal Consistency Failure SST Vector Service Routines
;
;  These routines are vectored on occurrence of a normally fatal error
;  during task execution.  This type of error usually occurs only on
;  a programming failure, so walkback and cleanup routines here are more
;  angled toward finding and fixing a problem than just getting out
;  cleanly.
;
;  Inputs:  Depends on trap executed
;
;  Outputs:  None
;
;  Register dispositions:  R0, R1, R2 modified
;
;  Variable dispositions:  SR0, SR1, SR2, SDSW, SSR0, SSR1, SSR2,
;                          SIOP modified


;  Odd address or nonexistent memory addressing error vector

SST000:    MOV     $DSW, SDSW              ; Save the DSW
   CPRINT  S35                     ; "-F-SC0, SST error - Odd address ..."
   BR      SSTDMP                  ; Go hit common dump code


;  Memory protect violation vector

SST001:    MOV     $DSW, SDSW              ; Save the DSW
   MOV     (SP)+, SSR1             ; Pop instruction backup register
   MOV     (SP)+, SSR2             ; Pop virtual PC of faulting instr'n
   MOV     (SP)+, SSR0             ; Pop memory protect status register
   CPRINT  S36                     ; "-F-SC1, SST error - Memory prot ..."
   BR      SSTDMP                  ; Go hit common dump code


;  T-bit trap or breakpoint trap (BPT) vector

SST002:    MOV     $DSW, SDSW              ; Save the DSW
   CPRINT  S37                     ; "-F-SC2, SST error - T-bit trap ..."
   BR      SSTDMP                  ; Go hit common dump code


;  I/O trap (IOT) vector

SST003:    MOV     $DSW, SDSW              ; Save the DSW
   CPRINT  S38                     ; "-F-SC3, SST error - I/O trap (IOT)"
```

```
        BR      SSTDMP                  ; Go hit common dump code


        ;  Reserved instruction vector

SST004: MOV     $DSW, SDSW                      ; Save the DSW
        CPRINT  S39                     ; "-F-SC4, SST error - Reserved ..."
        BR      SSTDMP                  ; Go hit common dump code


        ;  Non-RSX EMT vector

SST005: MOV     $DSW, SDSW                      ; Save the DSW
        MOV     (SP)+, SIOP             ; Pop instruction operand off stack
        ASR     SIOP                    ; Shift down one bit
        CPRINT  S40                     ; "-F-SC5, SST error - Non-RSX ..."
        BR      SSTDMP                  ; Go hit common dump code


        ;  TRAP vector

SST006: MOV     $DSW, SDSW                      ; Save the DSW
        MOV     (SP)+, SIOP             ; Pop instruction operand off stack
        ASR     SIOP                    ; Shift down one bit
        CPRINT  S41                     ; "-F-SC6, SST error - TRAP instruction"
        BR      SSTDMP                  ; Go hit common dump code


        ;  Floating point exception vector

SST007: MOV     $DSW, SDSW                      ; Save the DSW
        CPRINT  S42                     ; "-F-SC7, SST error - Floating ..."

        ;  Debugging dump section

SSTDMP: MOV     R0, SR0                         ; Save R0
        MOV     R1, SR1                 ; Save R1
        MOV     R2, SR2                 ; Save R2
        MOV     #S43A, R0               ; Load target field address
        MOV     (SP), R1                ; Load faulting PC for printing
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S43                     ; "     Program counter (PC):  xxxxxx"

        MOV     #S49A, R0               ; Load target field address
        MOV     (SP), R1                ; Load faulting PC for printing
        SUB     #SVTK, R1               ; Make the PC relative to the base
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S49                     ; "Program counter, MLCODE ..."

        MOV     #S44A, R0               ; Load target field address
        MOV     2(SP), R1               ; Load faulting PSW for printing
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S44                     ; "Processor status (PSW):  xxxxxx"
```

```
        MOV     #S51A, R0               ; Load target field address
        MOV     SDSW, R1                ; Load faulting $DSW for printing
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S51                     ; "     Directive status word ..."

        MOV     #S52A, R0               ; Load target field address
        MOV     SR0, R1                 ; Load faulting R0 for printing
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S52                     ; "     Processor general register ..."

        MOV     #S53A, R0               ; Load target field address
        MOV     SR1, R1                 ; Load faulting R1 for printing
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S53                     ; "     Processor general register ..."

        MOV     #S54A, R0               ; Load target field address
        MOV     SR2, R1                 ; Load faulting R2 for printing
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S54                     ; "     Processor general register ..."

        MOV     #S55A, R0               ; Load target field address
        MOV     R3, R1                  ; Load faulting R3 for printing
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S55                     ; "     Processor general register ..."

        MOV     #S56A, R0               ; Load target field address
        MOV     R4, R1                  ; Load faulting R4 for printing
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S56                     ; "     Processor general register ..."

        MOV     #S57A, R0               ; Load target field address
        MOV     R5, R1                  ; Load faulting R5 for printing
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S57                     ; "     Processor general register ..."

        MOV     #S45A, R0               ; Load target field address
        MOV     SSR0, R1                ; Load SR0 for printing
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S45                     ; "     Memory protect status ..."

        MOV     #S47A, R0               ; Load target field address
        MOV     SSR1, R1                ; Load faulting SR1 for printing
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S47                     ; "Instruction backup register ..."

        MOV     #S46A, R0               ; Load target field address
        MOV     SSR2, R1                ; Load faulting SR2 for printing
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S46                     ; "     "Virtual PC of faulting ..."

        MOV     #S50A, R0               ; Load target field address
        MOV     SSR2, R1                ; Load faulting SR2 for printing
        SUB     #SVTK, R1               ; Make the PC relative to the base
```

```
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S50                     ; "     "Virtual PC (SR2), SVCODE ..."

        MOV     #S48A, R0               ; Load target field address
        MOV     SIOP, R1                ; Load faulting operand for printing
        CALL    $CBOMG                  ; Convert binary to octal magnitude
        CPRINT  S48                     ; "     Instruction operand lower ..."

;   Wait 5 seconds for all I/O to clear, then exit with error status

        DIR$    #MARKTM                 ; Set a 5 second marktime
        DIR$    #WAITFR                 ; Wait for marktime to expire

        DIR$    #EXERR                  ; Exit with error status
```

Opening FCS Files By Wildcard From Your FORTRAN Program

Mark Chatterton
General Mills
9000 Plymouth Ave. N.
Minneapolis, MN 55427
(612)-540-3490

WHAT IT IS:

This article describes a series of FORTRAN callable FCS interface routines which allow a FORTRAN program to open files by wildcard. After specifying a wildcard string, the main program uses successive open and close operations to work its way through the series of files matching the wildcard specification. At any time, the program may also reset itself to start over at the beginning of the wildcard list.

WHY IT IS:

My department runs a DEC PDP-11 to do real-time applica- tions development. Unfortunately, the rest of building uses an IBM 4341 to do virtual-time data processing and office automation type things. One bit of software running on the IBM allows IBM users to exchange notes, phone messages, and insults via their CRTs (not

unlike an electronic mail system). For a time we were mercifully spared of this high-tech pest- ering, but eventually people demanded that we be included in the fun and games.

In order to accomplish note passing from the IBM to the DEC, we used DEC's 2780 emulator package. This package allows ASCII file transfers to and from IBM. When a new file arrives from the IBM, the software dequeues it, places it into a designated ac- count on the DEC machine, and names it PUNCH.something. You end up with an account full of files called PUNCH.001, PUNCH.002, etc. Some of these files are note files which must be copied over to the proper person on the DEC machine. To handle this, we wrote a receiver program that opens all the files of the form "PUNCH.*;*", checks to see if they are notes, and passes them along to the addressee.

The mainline code for the receiver was done in FORTRAN, which has no provisions for wildcard file specs. So, with F77DBG, USEROPEN, and FCS manual in hand, we groped about until we came up with some- thing that seemed to work.

HOW IT IS:

The routines I've included contain four entry points called WLDINI, WLDOPN, WLDRST, and WLDNAM. The following types of wild- cards are supported by these routines:

```
TEST.DAT;*
TEST.*;*
*.DAT;*
*.*;*
```

The procedures will not work with the following types:

```
*.DAT
TEST.*
*.*
```

Wildcard UIC's will not work. What follows is an explanation of what each subroutine does and how to call them:

WLDINI:

Use this routine to set up the wildcard name string you want. The calling sequence is:

```
CALL WLDINI (DEV,DEVSZ,UIC,UICSZ,FIL,FILSZ)
```

Where:

DEV is the ASCII device name, like 'DL3:' or 'DK2:'.
DEVSZ is the length in bytes of the DEV string.

UIC is the ASCII UIC string, like '[5,4]' or '[1,2]'.
UICSZ is the length in bytes of the UIC string.
FIL is the ASCII file name, like 'TEST.DAT;*' or '*.DAT;*'.
FILSZ is the length in bytes of the FIL string.

If you leave out any of these fields, the standard defaults will take effect.

Once you have set up the name using WLDINI, you are ready to do an open operation using WLDOPN.

Examples:

```
CALL WLDINI ('DL2:',4,'[1,2]',5,'*.CMD;*',7)
     Sets up to open all .CMD files in [1,2],on DL2:.

CALL WLDINI (,,,,'*.CMD;*',7)
     Sets up to open all .CMD files in your current account
     on your SY: disk.
```

WLDOPN:

WLDOPN will be used as a USEROPEN routine in your OPEN statement. It will take control from the FORTRAN OTS, find the next file that matches the string you set up with WLDINI, and open the file for read/write.

WLDOPN is an external procedure, so it must be declared in an EXTERNAL statement at the beginning of your program.

An important point: FORTRAN syntax requires you to include a file name in your OPEN statement using the FILE= keyword. It doesn't matter what you put here, since WLDOPN will use the string set up by WLDINI. The file name you use does not even have to exist. HOWEVER - FORTRAN will assign LUNs according to the name it finds in the FILE= keyword. It is reccommended, therefore, that you at least give FORTRAN the proper device in your FILE= keyword. In this way the OTS will assign the LUN correctly and you'll have one less thing to worry about.

The next time you execute the same OPEN statement, you will get the next file that matches your wildcard string and so on. When there are finally no more such files left, the next OPEN command will return a No-Such-File error to the OTS.

I have included an example of all this at the end of this section.

WLDRST:

Use WLDRST to reset your program back to the beginning of the list of wildcard files at any time. The next time you do an OPEN after calling WLDRST, you will start over at the top of the series of files that matches your wildcard string. You get the same effect by calling WLDINI again, but with more overhead.

WLDNAM:

Use this routine to find out the actual name of the currently open file. Calling sequence is:

CALL WLDNAM (DEVSTR,DEVNUM,FILNAM,FILTYP,VER)

Where:

DEVSTR = Two byte buffer to receive ASCII device name.
DEVNUM = One word buffer to recieve binary unit number.
FILNAM = Three word buffer to receive Rad-50 file name.
FILTYP = One word buffer to receive Rad-50 file type.
VER = One word buffer to receive binary version number.

You may leave out whatever fields you aren't interested in.

EXAMPLE PROGRAM

This sample program will open all .DAT files in the current account on DL3: and print the names of the files on the screen. It will then do a reset, and reopen the files again, this time printing the first line of each file onto the screen.

```
C*************************************************************************
C
   Program Sample
C
C  Instructions for building:
C
C  >F77 SAMPLE=SAMPLE
C  >MAC WLDOPN=WLDOPN
C  >TKB SAMPLE/CP/FP=SAMPLE,WLDOPN,LB:[1,1]F770TS/LB
C
C  Declare wildcard character strings. NAME will be used as the filename
C  in our OPEN statement. The equivalence is used so that FORTRAN will
C  assign the LUN to the correct device for us.
C
   Character Dev*4, Fil*7, Name*20
   Equivalence (Dev, Name)
   Integer DevSz, FilSz

   Data Dev, DevSz /'DL3:',4/                    !Device to look on.
   Data Fil, FilSz /'*.DAT;*',7/                 !Wildcard string.
   Data Name(5:) /'BOGUS.FIL'/                   !Bogus name for FORTRAN
```

```
C
C  Set up buffers for WldNam call and Rad50 conversion.
C
   Integer RelFil(3), RelTyp, RelVer
   Character*3 ChrFil(4)
C
C  Define input buffer for read operation
C
   Character InLine*80
C
C  Declare WLDOPN as external
C
   External WldOpn
C***
C
C  Start executable code.
C
C  Trap No-Such-File errors
C
   Call Errset (29,.True.,.False.,.True.,.False.)
C
C  Use WLDINI to set up the wildcard string.
C
   Call WldIni (Dev,DevSz,,,Fil,FilSz)
C
C  Open next file in line
C
50 Open (Unit=1, File=Name, Status='Old', UserOpen=WldOpn, Err=500)
C
C  Find out what file we've actually opened, and print it to TI:
C
   Call WldNam (,,RelFil,RelTyp,RelVer)            !Get file name info

   Call R50Asc (3,RelFil(1),ChrFil(1))             !Translate file name
   Call R50Asc (3,RelFil(2),ChrFil(2))             !   from Rad50
   Call R50Asc (3,RelFil(3),ChrFil(3))             !   to ASCII
   Call R50Asc (3,RelTyp,ChrFil(4))                !Do file type too.

   Write (5,100) ChrFil, RelVer                    !Write info to screen
100        Format (X,3A,'.',A,';',I3)
C
C  Now close the file and try for the next one
C
   Close (1)
   Goto 50
C***
C
C  If we get here we had an open error. Error #29 (no such file)
C  is OK (just means we got to end of wildcard list).
C  Exit on other errors.
C
500        Call ErrSns (I)                         !Retreive error number
   If (I .Ne. 29)  Stop '- Open error on file.'
```

```
   Write (5,*) '*** Done with pass one ***'
C
C  Reset the file pointer to top of wildcard list
C
   Call WldRst
C
C  Open next file in line
C
600        Open (Unit=1, File=Name, Status='Old', UserOpen=WldOpn, Err=1000)
C
C  Read first line in the file and print it on screen.
C
   Read (1,650)  InLine                        !Read the first line
650        Format (A)
   Write (5,*) InLine                          !Write it to screen
C
C  Now close the file and try for the next one
C
   Close (1)
   Goto 600
C***
C
C  If we get here we had an open error. Error #29 (no such file)
C  is OK (just means we got to end of wildcard list). Exit on other errors.
C
1000       Call ErrSns (I)                          !Retreive error number
   If (I .Ne. 29)  Stop '- Open error on file.
   Write (5,*) '*** Done with pass two ***'
   Call Exit
   End
```

HERE IT IS:

What follows is the source code for the wildcard open routines. Good luck in using them, and I hope they turn out as useful for you as they have for us.

```
   .TITLE  WLDOPN

   .LIST   TTM
   .NLIST  SEQ,BIN

   .MCALL  EXIT$S,FDOF$L,QIOW$S,OFID$U

   FDOF$L                                      ;Define FDB offsets

DEVICE:    .BLKW   3                           ;Device name
DIRECT:    .BLKW   5                           ;Search directory
FILNAM:    .BLKW   9                           ;File name here
```

```
DS:         .BLKW   6                       ;Dataset descriptor

FLAG:       .BLKW   1                       ;First time flag
STAT:       .BLKW   1                       ;Saved status
NEXT:       .BLKW   1                       ;Saved pointer to next file

FDBADD:     .BLKW   1                       ;Saved pointer to fdb

WLDINI::
   MOV      #DS,R0                          ;Get addres of dataset desc.
   MOV      #6,R1                           ;Loop counter
CLEAR:      CLR     (R0)+                           ;Clear Dataset desc.
   SOB      R1,CLEAR                        ;Until done

DEV:        CMP     #1,@R5                          ;Done yet ?
   BGT      INIDUN                          ;Branch if not
   CMP      #-1,2(R5)                       ;Device name present ?
   BEQ      UIC                             ;Branch if yes
   MOV      #DEVICE,R0                      ;Destination address
   MOV      #DEVICE,DS+2                    ;Address into DS
   MOV      2(R5),R1                        ;Source address
   MOV      @4(R5),R2                       ;Number of bytes to move
   MOV      R2,DS                           ;Length into DS
DEV1:       MOVB    (R1)+,(R0)+                     ;Copy next byte
   SOB      R2,DEV1                         ;Until done

UIC:        CMP     #3,@R5                          ;Done yet ?
   BGT      INIDUN                          ;Branch if not
   CMP      #-1,6(R5)                       ;UIC present ?
   BEQ      FIL                             ;Branch if yes
   MOV      #DIRECT,R0                      ;Destination address
   MOV      #DIRECT,DS+6                    ;Address into DS
   MOV      6(R5),R1                        ;Source address
   MOV      @10(R5),R2                      ;Number of bytes to move
   MOV      R2,DS+4                         ;Length into DS
UIC1:       MOVB    (R1)+,(R0)+                     ;Copy next byte
   SOB      R2,UIC1                         ;Until done

FIL:        CMP     #5,@R5                          ;Done yet ?
   BGT      INIDUN                          ;Branch if not
   CMP      #-1,12(R5)                      ;File name present ?
   BEQ      INIDUN                          ;Branch if yes
   MOV      #FILNAM,R0                      ;Destination address
   MOV      #FILNAM,DS+12                   ;Address into DS
   MOV      12(R5),R1                       ;Source address
   MOV      @14(R5),R2                      ;Number of bytes to move
   MOV      R2,DS+10                        ;Length into DS
FIL1:       MOVB    (R1)+,(R0)+                     ;Copy next byte
   SOB      R2,FIL1                         ;Until done

INIDUN:     CLR     FLAG            ;Clear first time through flag
   RETURN
```

```
WLDOPN::
   MOV     2(R5),R0                    ;Get FDB address
   MOV     R0,FDBADD                   ;Save it for WLDNAM routine
   MOV     R0,R1                       ;Copy FDB address
   ADD     #F.FNB,R1                   ;Get name block address
   MOV     #DS,R2                      ;Point to DS
   CLR     R3                          ;Clear pointer to default
                                       ;   file name block

   CALL    .PARSE                      ;Set up file name stuff

   TST     FLAG                        ;First time through ?
   BEQ     FIND                        ;Branch if yes

   MOV     STAT,N.STAT(R1)             ;Restore status
   MOV     NEXT,N.NEXT(R1)             ;Restore pointer to next file

FIND:      INC     FLAG                        ;Indicate not first
   CALL    .FIND                       ;Find the file
   BCS     DONE                        ;IE.NSF indicates end

   OFID$U  R0                          ;Open file for processing

DONE:      MOV     F.ERR(R0),R0                ;Return error code for main

   MOV     N.STAT(R1),STAT             ;Save status
   MOV     N.NEXT(R1),NEXT             ;Save pointer to next file
   CLR     N.STAT(R1)                  ;Clear these two fields
   CLR     N.NEXT(R1)                  ;So FCS doesn't get confused.

   RETURN                              ;Return to main routine

WLDRST::
   CLR     FLAG                        ;Clear first time flag
   RETURN

WLDNAM::
   MOV     FDBADD,R0                   ;Get FDB address
   ADD     #F.FNB,R0                   ;Point to file name block

   CMP     #1,@R5                      ;Done yet ?
   BGT     NAMDUN                      ;Branch if yes
   CMP     #-1,2(R5)                   ;Want device name ?
   BEQ     DEVNUM                      ;Branch if not
   MOV     2(R5),R1                    ;Get device name buffer add.
   MOV     N.DVNM(R0),(R1)             ;Copy device name to user

DEVNUM:    CMP     #2,@R5                      ;Done yet ?
   BGT     NAMDUN                      ;Branch if yes
   CMP     #-1,4(R5)                   ;Want device number ?
   BEQ     FLNAME                      ;Branch if not
   MOV     4(R5),R1                    ;GEt device # buffer add.
   MOV     N.UNIT(R0),(R1)             ;Copy device # to user
```

```
FLNAME:   CMP     #3,@R5          ;Done yet ?
   BGT    NAMDUN                  ;Branch if yes
   CMP    #-1,6(R5)               ;Want file name ?
   BEQ    FILTYP                  ;Branch if not
   MOV    6(R5),R1                ;Get file name buffer addr
   MOV    R0,R2                   ;Copy FNB addr into R2
   ADD    #N.FNAM,R2              ;Point R2 to file name info
   MOV    (R2)+,(R1)+             ;Copy 1st word
   MOV    (R2)+,(R1)+             ;Copy 2nd word
   MOV    (R2),(R1)               ;Copy 3rd word

FILTYP:   CMP     #4,@R5          ;Done yet ?
   BGT    NAMDUN                  ;Branch if yes
   CMP    #-1,10(R5)              ;Want the file type ?
   BEQ    VER                     ;Branch if not
   MOV    10(R5),R1               ;Get file type buffer addr.
   MOV    N.FTYP(R0),(R1)         ;Copy type to user

VER:      CMP     #5,@R5          ;Done yet ?
   BGT    NAMDUN                  ;Branch if yes
   CMP    #-1,12(R5)              ;Want the version ?
   BEQ    NAMDUN                  ;Branch if not
   MOV    12(R5),R1               ;Get version buffer addr
   MOV    N.FVER(R0),(R1)         ;Copy version to user

NAMDUN:   RETURN
   .END
```

MULTITASKER

RSX Menu Item Submission Form

Name:_____ Firm:_____

Address:_____ Phone:_____

_____

_____

Operating System: RSX-11M RSX-11M+ Micro-RSX VAX-RSX POS-RSX

Describe the capability you would like to see available. Be as specific as possible. Please don't assume we know how it's done on the XYZ system. Explain how the capability would be useful and give an example of its use. If you wish, suggest a possible implementation of your request

Return forms to: Allen Jay Bennett
State Systems, Inc.
2004 Inverway Ct.
Kalamazoo, Michigan 49003

Printed in the U.S.A.

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

STATUS CHANGE

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

( ) Change of Address
( ) Please Delete My Membership Record
(I Do Not Wish To Remain A Member)

DECUS Membership No: _____

Name: _____

Company: _____

Address: _____

_____

_____

State/Country: _____

Zip/Postal Code: _____

Mail to: **DECUS - ATTN: Subscription Service**
249 Northboro Road, BP02
Marlboro, Mass.

Affix mailing label here. If label is not available, print old address here. Include name of installation, company, university, etc.