MEMO TEN-SYS 9

To: BBN TEN-SYS Group

From: D. Murphy

Subj: Monitor Calls and Pseudo-Interrupts

Date: April 7, 1969

- - - - - - - - - - - - - - -

There are two types of monitor calls, UUO's and JSYS's. There are two classes of each of these, "fast" and "slow". "Slow" means that because of some additional overhead, the routine may be pseudo-interrupted and subsequently resumed. "Fast" means that the call will take sufficiently little time that pseudo-interrupt requests may be deferred until completion of the call, and that the additional overhead is undesirable and unnecessary.

The two classes of UUO's are distinguished by a bit in the left half of the UUO dispatch word. That is, if a UUO with opcode n is "slow", then

```
UUOT+n/   XWD Ø,ADR
```

and if it is "fast", then

```
UUOT+n/   XWD 4ØØØØØ,ADR
```

If the bit is off, the UUO dispatcher will go to the UUO code via the "slow-call" setup routine (shown later), whereas control will be transferred directly to the UUO code for a "fast" UUO.

The two classes of JSYS are distinguished by virtue of the fact that the "slow" JSYS code contains an explicit call to the "slow-call" setup routine, whereas the code for a fast JSYS does not.

Entry Procedure

The "slow-call" setup routine, called MENTR (MONITOR ENTER), is invoked from monitor code by execution of the instruction JSYS MENTR. Note that a user-mode program cannot execute this instruction with the same result because the effective address is greater than 1ØØØ(octal).

The following convention is observed:

> All user-executable monitor call instructions (JSYS and UUO) store their return PC in the same

cell. It is called FPC and is located in the TS block.

This is necessary to insure correct action on pseudo-interrupt requests occurring during the execution of monitor code as will be demonstrated.

The "slow-call" setup routine maintains a stack containing returns and temp storage for slow monitor routines. This stack is also used to hold the original user AC's during the course of execution of a monitor routine so that they may be restored if the call is aborted. When executing a user-to-monitor call, this setup routine places the push pointer in AC17, and saves the user AC's and the return. When executing a monitor-to-monitor call, the push pointer is assumed to already exist in AC17 and the AC's need not be saved, so the MENTR routine need only add the return to the stack (ala PUSHJ).

Returns

The return from a fast JSYS or UUO would appear to be simply JRSTF @FPC, and it would be, except for the requirments Of the pseudo-interrupt logic. The return from a slow monitor call is effected by the return routine MRETN (MONITOR RETURN) which performs the inverse function of MENTR.

Pseudo-Interrupts

Pseudo-interrupt requests can occur at any time. A PSI (pseudo-interrupt) request may be processed immediately if it occurs while the process is in user mode. When the interrupt request occurrs during a monitor call however, it may be serviced immediately only if it can be guaranteed that:

1. Temp storage, including PC and AC's, in use by the interrupted call is protected from change by the user directly or by other monitor calls executed in the user's interrupt service routine. Otherwise, the routine may malfunction on being resumed, and, since it is running in monitor mode, could possibly destroy the monitor.

2. The routine can be aborted (by explicit user request) without leaving anything in inconsistent or transitory states.

Sometimes these conditions can not be met, so a PSI request must be saved and serviced at a later time.

Interruptibility of "Slow" Monitor Calls

In order to meet condition 1 above, it is at least necessary that the temp storage in use at the time of the interrupt be identifyable. The most convenient way to do this is to establish a stack (push list) in the TS block to be used for all temp storage for all interruptable monitor routines. This stack, then, along with the AC's and the process PC would represent the complete state of the process. When a PSI is requested, the AC's and PC are added to the stack and the stack pointer increased accordingly. This procedure effectively protects temp storage as required by condition 1. Additional monitor calls can be entered from the user's interrupt service routine, and additional interrupts can be initiated on higher priority channels to a depth limited only by the size of the stack and the number of priority levels.

The stack provides one other function, that of saving the contents of the user program's AC's at the time of the monitor call. This is necessary in order to be able to abort the monitor routine and allow it to be restarted in the case that the process (fork) is terminated or the environment is saved. Also, since the interrupt PC given to the user program on an interrupt from a monitor call will point to the monitor call instruction (not to the monitor address where the routine actually stopped), it is desirable that the AC's given to the user likewise be those at the time of the call.

The routines MENTR and MRETN handle the maintenance of the stack on entering and leaving "slow" monitor routines as mentioned above.

Fast-Slow Distinction

As can be seen, there is an overhead involved in the stack maintenance procedure, a cost greater than that of a simple JSYS-JRSTF call and return sequence. However, monitor routines which are so short that this overhead time is a significant fraction of their execution time are likewise so short that there is no problem in deferring interrupt service to their completion. This is precisely the distinction between "fast" and "slow" monitor calls.

Interrupting "Fast" Monitor Calls

Since fast monitor routines are by definition not in a state to be interrupted, it must be possible to save an interrupt request and service it at a later time, preferably at the termination of the fast routine. We propose to do

this by making the return for fast monitor calls be done by executing the instruction XCT MJRSTF. The contents of MJRSTF will normally be JRSTF @FPC if there has been no pseudo-interrupt request. Since all returns are saved in FPC, this instruction is always the appropriate one. If there was a PSI request, the monitor's PSI control routine will have changed the contents of MJRSTF to JRST PSISVØ which will again consider initiating an interrupt, assuming now that the process PC is specified by the contents of FPC.

PSI Strategy

The process by which the PSI routine decides whether to interrupt "immediate" or "deferred" is somewhat complex. The first decision factor is the state of the user mode flip-flop available in the process PC word. If the process to be interrupted is in user mode, the interrupt can be done immediately. If the process is in monitor mode, several cases must be distinguished, the most obvious of which is "fast" vs. "slow" code. The flag SLOWF (in the TS block) makes this distinction. It is initialized to -1; entering slow monitor code (via MENTR) makes it positive, leaving returns it to its previous state. Therefore if SLOWF is negative (and process is in monitor mode), "fast" code is implied and interrupt request is deferred as described above.

One other flag, INTDF (INTERRUPT DEFER FLAG) is also included to enable "slow" routines to temporarily defer interrupts when aborting the routine would leave something in an inconsistent state (e.g. during a change to the PAC slot list structure). It is also necessary for some of the monitor-to-monitor calling sequences shown later. This flag is normally -1 (off, i.e. interrupts not deferred). It is turned on with AOS, and turned off with XCT INTDFF. INTDFF normally contains SOS INTDF if no interrupt is waiting, otherwise JSYS PSISV1.

Note that the routines at PSISVØ and PSISV1 do not necessarily initiate the interrupt whenever they are entered, rather they reconsider the state of the process as specified by the various flags and accordingly either initiate the interrupt or set up another defer trap and resume the sequence.

Sample Routines

Following are the actual routines used for entering and leaving slow JSYS's and for decoding UUO's.

```
;UUO DISPATCH ROUTINE

41/     JSYS UUOH               ;JSYS RATHER THAN JSR TO BE REENTRANT

UUOH:   XWD FPC,.+1             ;RETURN GOES TO FPC AS FOR JSYS'S

        MOVEM 1,XMENTR          ;AC1 => TEMP
        LDB 1,[POINT 9,4Ø,8]    ;GET OP CODE
        CAIL 1,1ØØ              ;CHECK FOR OUT OF BOUNDS
        JRST ITRAP              ; ILLEGAL INSTRUCTION
        SKIPG 1,UUOT(1)         ;GET DISPATCH WORD AND CHECK FAST OR SLOW
        JRST UUOH2              ; FAST...
        EXCH 1,XMENTR           ;SLOW, RESTORE AC1, SETUP DISPATCH ADR
        JRST UUOH1              ;TO SLOW-CALL SETUP ROUTINE

UUOH2:  EXCH 1,XMENTR           ;FAST, RESTORE AC1, SETUP DISPATCH ADR
        JRST @XMENTR            ;DIRECTLY TO ROUTINE
```

Comments:

At UUOT is a 1ØØ(octal) word dispatch table for UUO's with indicator bit in each left half as mentioned.

Slow UUO's exit with JRST MRETN, fast UUO's exit with XCT MJRSTF.

```
;SLOW-CALL SETUP ROUTINE

MENTR: XWD XMENTR,.+1          ;JSYS DISPATCH

;SLOW UUO'S ENTER HERE

UUOH1: EXCH Ø,FPC              ;GET RETURN PC
       TLNE Ø,UMODF            ;USER OR MONITOR MODE
       JRST MENT1              ; USER...
       PUSH 17,Ø               ;MONITOR,  RETURN => STACK
MENT2: MOVE Ø,XMENTR           ;LOCAL RETURN => FPC
       EXCH Ø,FPC              ;USERS ACØ => ACØ
       AOS SLOWF               ;INDICATE SLOW CODE
       XCT MJRSTF              ;JRSTF @FPC OR INTERRUPT

MENT1: MOVEM Ø,XMENT1          ;RETURN => TEMP
       MOVE Ø,17               ;SAVE USER'S AC17
       MOVE 17,UPP             ;SETUP PDL POINTER
       PUSH 17,XMENT1          ;SAVE USER RETURN FOR POSS. PSI
       PUSH 17,Ø               ;USER'S AC17 => STACK
       PUSH 17,FPC             ;USER'S ACØ => STACK
       MOVSI Ø,1               ;USER'S AC'S 1 TO NSAC => STACK
       HRRI Ø,1(17)            ; FOR POSSIBLE PSI
       ADD 17,[XWD NSAC-1,NSAC-1]
       BLT Ø,Ø(17)
       PUSH 17,XMENT1          ;RETURN => STACK
       JRST MENT2
```

Comments:

Return is always last entry on stack, so skip return can be done by AOS Ø(17), etc.

When coming from user mode, additional procedure is to setup stack pointer, save AC's and original return (in case regular return is modified).

Interrupt requests occurring during this code will be deferred (to MENT2+3) if entry is from user mode, may be immediate or deferred depending on other flags if entry is from monitor mode.

```
;SLOW-CALL RETURN ROUTINE

MRETN: SOS SLOWF              ;LEAVING (ONE LEVEL OF) SLOW CODE
       MOVEM Ø,FPC            ;SAVE ACØ
       POP 17,Ø               ;POP RETURN
       TLNE Ø,UMODF           ;USER OR MONITOR MODE
       MOVE 17,-NSAC(17)      ;USER, RESTORE USER'S AC17
       EXCH Ø,FPC             ;RESTORE ACØ, SETUP RETURN
       XCT MJRSTF             ;RETURN OR INTERRUPT
```

Comments:

This routine is like a POPJ with flag restoring

When returning to user mode, actual AC's (not AC's saved on entry) except 17 are returned.

Interrupt requests occurring during this code will be deferred (to MRETN+1) or immediate as for MENTR.

Initiation of Interrupt

When an interrupt is to be initiated from a monitor call, these TS block cells must be added to the stack:

UPP - Initial stack pointer; changed only for interrupt service (monitor to user transfer), set to current stack position at start of interrupt

4Ø - General UUO temp

SLOWF - Slow code level (flag)

FPC - Temp possibly is use by MENTR or MRETN
XMENTR - " "
XMENT1 - " "

Also, the following must be added to the stack:

AC's Ø-NSAC - presumed to be in use by mon code

process PC - pointing into monitor routine

Then:

1. Current stack pointer => UPP
2. Get original user AC's and PC
3. Go to user's interrupt routine

When the user debreaks, the monitor routine will be resumed if the user did not change the interrupt PC, otherwise the stack will be cleared back one level (using the saved UPP), and the process will be started in user mode at the specified location.

Nested Monitor Calls

Monitor calls may be executed within other monitor calls, but extra instructions are required in some cases. There are four possibilities:

Slow to Slow

Same as user; save 40 if nested UUO

Slow to Fast

Become non-interruptable first, i.e.

```
AOS INTDF               ;DEFER INTERRUPTS
one or more fast calls
XCT INTDFF              ;SOS OR JRST
```

Fast to Fast

Save return on special stack, i.e.

```
MOVE AC,FPC
AOS FPTR                ;SPECIAL STACK POINTER
MOVEM AC,@FPTR
one or more fast calls
MOVE AC,@FPTR
SOS FPTR
MOVEM AC,FPC
```

Fast to Slow (Arising where fast routine wants to be conditionally slow)

Execute slow-routine entry procedure and observe slow routine conventions.

```
AOS INTDF               ;DEFER INTERRUPTS
JSYS MENTR              ;INITIALIZE STACK, ETC.
PUSH 17,..              ;SAVE LOCAL TEMPS
..                      ..
XCT INTDFF              ;ENABLE INTERRUPTS.
```

The routine is now effectively "slow", and should return with JRST MRETN. It can become "fast" again with the following cludge:

```
AOS INTDF               ;DEFER INTERRUPTS
POP 17,..               ;RESTORE LOCAL TEMPS
..                      ..
POP 17,TAC              ;ORIGINAL RETURN
PUSHJ 17,MRETN          ;UNDO SLOW SETUP AND RETURN HERE
```

```
        MOVEM TAC,FPC           ;REPLACE ORIGINAL RETURN
        XCT INTDFF              ;ENABLE INTERRUPTS
```

This last case should be done only rarely and with extreme caution to be sure that there is not a higher level fast routine (in which this one is nested) which does not expect to be interrupted and which may have vulnerable temps.

4/5/69 DLM