TEN-SYS-7

M E M O R A N D U M

To: TEN-SYS-GROUP

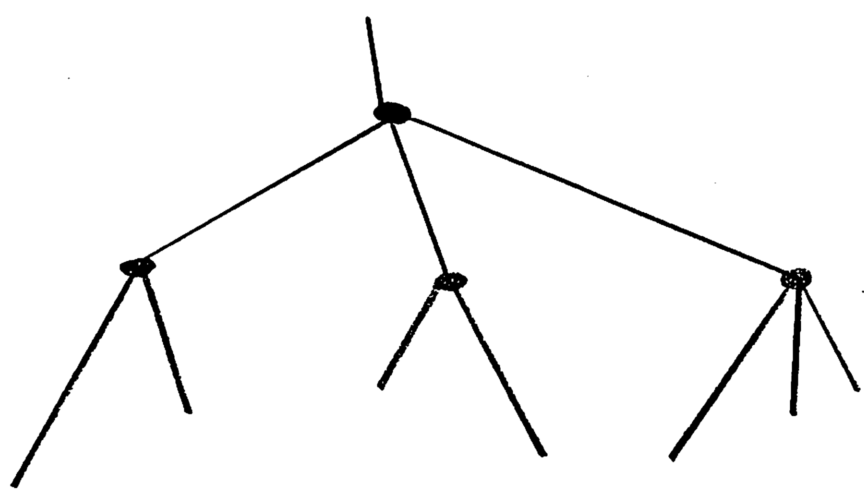
From: Ted R. Strollo

Subject: TEN-SYS FORK AND PSEUDO-INTERRUPT STRUCTURE

Date: April 2, 1969

Fork Structure and Communication

The BBN PDP-1Ø Ten-SYS Software will permit each job to have multiple simultaneously runable processes or forks. The fork structure will be quite similar to the SDS-94Ø structure in that both parallel and subsidiary forks will be allowed. The structure will look like an inverted tree.



It will be possible for a fork to create either parallel or inferior (subsidiary) forks but not superior forks in the structure. A fork can communicate with other members of the structure by

(a) sharing memory

(b) termination, initiatior, or suspension of any parallel or subsidiary fork.

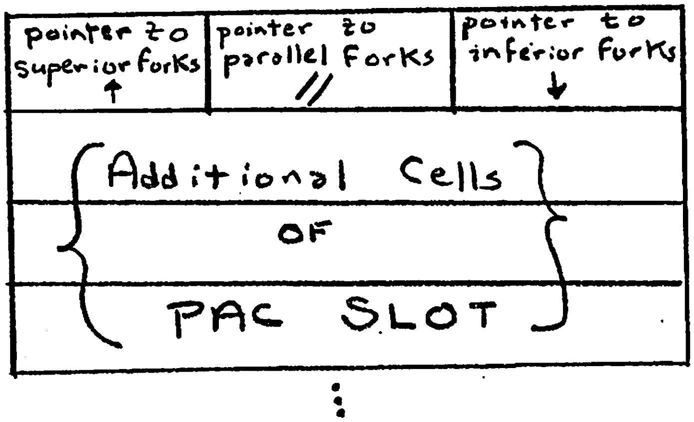
(c) pseudo (software simulated) interrupts

Fork Accumulators

The accumulator values for a fork which is being initiated, has been suspended, or has been terminated will be found in virtual core locations Ø - 17_8 of the fork's address space. The accumulator values of the "running" process will be saved in the process TS block. This is necessary because the user has access to virtual core locations Ø - 17_8 by relabelling fork address space, and the system cannot permit the user to modify the accumulator values of a fork dismissed in the midst of running exec mode code.

Fork Structure Specification

The fork structure will be specified by a pointer structure in the primary temporary storage block. This pointer structure is quite similar to the SDS-94Ø PAC slot structure. A 12 bit value defines a fork of this job. The fork specification will expand to another full page if the allocated area in the primary temporary storage block becomes full. The 12 bit fork value is an address of a work in either the primary temporary storage block or the extension page.



When a process initiates a fork, the monitor returns to the initiator the forks pointer number which can be used by the initiator to reference this fork in the future. Any process can reference another fork by using its pointer number as a handle, a monitor call will also translate a relative fork reference (such as the left-most fork two levels inferior to me) to a pointer number.

<u>Fork Suspension</u>

A process may be suspended (temporarily stopped) by one of several conditions

(1) Type-in of super-panic (C^c)

(2) The process's execution of an instruction which causes a hardware alarm (memory trap, instruction trap, etc.)

(3) The request for suspension from a parallel or superior process.

(4) Pseudo-Interruption of a process

The system will <u>never</u> permit an environment to be saved which has processes suspended during the midst of execution of JSYS code. The Mini System will not suspend a process in the midst of JSYS execution but will instead either run the

JSYS to completion or abort it so that it will be restorted from scratch (all AC's preserved) if the process is ever resumed. The exact mechanism for deciding whether to complete or abort a JSYS has not been specified, but it is clear that the mechanism will complete JSYS's which would complete in the near future and abort those which would take indefinite or long periods of time.

Since the hardware does not have a user mode transition trap, and we do not intend to simulate this trap in all exits from monitor calls, the exit location from the top level (user called) JSYS will be changed and flagged on any deferred action fork suspension. All code in the JSYS which uses this return or changes it (with the single exception of the JRST@ to return from the JSYS call) must first check this flag to see if the return has been modified. If the flag is set, the actual return will have been saved in an alternate location. Note that these restrictions on fork suspension in no way affect the reschedulability of other processes during the execution of JSYS code.

JSYS's will occasionally reach code which is non-terminable for this particular process. Again such a specification is generally independent of reschedulability butthe code must be complete sometime for the specified process. An example of such a non-terminable sequence would be making changes to the PAC slot list (pointer) structure which would have chaotic consequences if it were terminated mid-stream. When a fork suspension request is received during the execution of non-terminable code it is always saved away and reconsidered when a JSYS goes through the subroutine of change from non-terminable to terminable. Note that terminablity must be a fork temporary variable and must be a counter which is decremented for every reason for non-terminablity and incremented (then checked for Ø) whenever a particular condition for non-terminability is lifted. This will permit arbitrary JSYS's to be called from non=terminable JSYS code.

Pseudo-Interrupts

Several conditions will cause the automatic suspension of one or more processes and the continuation of that process at specified locations called the pseudo-interrupt routines. Prior to the continuation, the process PC will have been saved so that the pseudo-interrupt routine may resume the process upon completion of its tasks. These conditions are:

(1) Terminal Pseudo Interrupts which are generated when selected terminal keys are typed.

(2) Illegal Instruction Traps (such as attempts to execute I/O instructions in ordinary User mode) or attempts to execute privileged monitor calls.

(3) Memory Traps including non-ex mem R.W.X traps and directed traps

(4) Arithmetic Processor Traps

(5) Unusual File Conditions (EOF, errors

(6) Specific Time of Day reached

(7) Generated Pseudo-Interrupts

(8) Fork Termination

(9) System Resource Allocation traps

Any number of these conditions can be assigned to one of 8 pseudo-priority levels. Up to 8 interrupts can be in progress simultaneously. Only high level (lower priority numbers) interrupts can interrupt a lower level pseudo-interrupt routine. It is necessary to exit a pseudo-interrupt routine via a monitor call which will reset the interrupt in progress status of a pseudo-priority interrupt. When a request comes along on a priority channel which has an interrupt in progress or which is lower in priority than any interrupt in progress, the request condition is remembered. (Successive requests for the same condition with the priority channel in this state will not cause additional interrupts.)

The user can turn the pseudo-interrupt system on or off. When the system is off, interrupt requests are remembered and will take when the system is turned back on. The user can also clear the entire interrupt system thereby forgetting all stacked requests.

There are approximately 72 possible pseudo-interrupt conditions; 36 of these correspond to the terminal pseudo interrupt keys, the rest are the various memory trap conditions, etc. Each condition specifies a particular location (42_8 + condition #) in the users virtual address space. This location contains a pointer to the pseudo-interrupt subroutine in the right half and a pseudo-priority number (Ø - 7_8) in the left half. Note the interrupted PC is not kept in the users address space but is accessible via a monitor call. The PAC slot in the primary TS block contains two words for each fork to indicate pseudo-interrupt condition arming. Each bit corresponds to one of the 72 conditions and if set means the condition is armed. The TS block for each process contains two words with a bit for each condition to remember a deferred request for a pseudo-interrupt (deferred by high priority request or user pseudo-interrupt channels off specification). There is also an 8 bit byte with a bit for each priority level to specify a pseudo-interrupt in progress on a priority level.

Pseudo Interrupt Fork Specification

When a particular pseudo-interrupt condition arises several forks may be suspended (or even terminated) and (generally) one fork will be pseudo-interrupted. It is not often straight forward to determing which fork should be interrupted. For example, when a terminal pseudo-interrupt character is typed, it is quite possible that several forks may be armed for that pseudo-interrupt condition none of which may be running.

The following rules specify which fork gets the various pseudo -interrupts.

Terminal Pseudo-Interrupts

Up to 36 terminal keys may be used to specify pseudo-interrupts. Each of these may be armed in multiple forks, but when a fork arms a particular key the assignment of that key passes to that fork alone. When that fork terminates or disarms the key, it will be passed up the fork structure in a parallel/superior fork search to the nearest fork which has the key armed. This will be implemented by having for each terminal a total of 36 (maximum) 12 bit bytes which are dispatched into by key (via an ASC II to one of 36 translation table). The 12 bit byte thus addressed is the fork number of the fork currently assigned to the key. Each terminal will commit only 2 words (6-12 bytes) for this purpose. The most commonly used 6 keys will be specified as the first 6 in this ASC II to one of 36 translation table. If any of the other keys are armed, an extension table will be used from a pool of dynamically assigned monitor storage.

Directed Pseudo-Interrupts

The generated pseudo-interrupts can be directed to a specific fork which completely specifies the fork to pseudo interrupt. The timer interrupts have the fork number pre-specified when the request is made for such an interrupt. (e.g. a fork might execute a monitor call requesting a timer interrupt of itself 1Ø seconds from now).

Non-Directed Pseudo-Interrupts Initiated Within a Fork

The attempted execution of an illegal instruction, attempted memory out of bounds reference, generated (non-directed) pseudo interrupt request, etc. are made while a particular fork is running and executing code. The fork which gets such a pseudo-interrupt is determined by scanning parallel and up the fork structure starting at the current fork or the nearest fork with the pseudo-interrupt armed. All forks below this interrupted fork are suspended in most cases (except for generated pseudo interrupts and AP traps under certain conditions).

AP Trap Special Conditions

The four conditions which cause the arithmetic overflow AP trap may be individually ignored in a particular process. If all 4 are to be ignorged, the monitor will be smart enough to disable this AP trap from unnecessarily degrading the operation speed of the running process.