A proposal for the values of a call stack entry and their manipulation by the return operations.

I) variables maintained in a stack entry

- p-counter (PC)
- p-counter modifier (PCM)
- F-return count (FRCN)

clerk code for our process at top of stack
put
Top of stack clerk code

This proposal only concerns the first 3.

A) The p-counter contains a non-negative integer
B) The p-counter modifier has one of 3 values
   i) about to execute instruction at address PC
   ii) in the middle of a complicated XT instruction
       at address PC
   iii) almost finished with a complicated XT instruction
        at address PC
C) The F-return count contains a non-negative integer
II) variables maintained in a sub process descriptor:

- BIT 0: interrupts armed
- BIT 1: interrupts inhibited
- BIT 2: interrupt waiting
- D: interrupt datum
- D: error selection mask
III) Fancy return instruction

a) parameters as follows:

```
P10 0:     Key
P2 0:     data params being returned, if any
P3 0C:    capability params being returned, if any
P4 0:     error code, if any
P5 0:     error number, if any
P6 0:     internal return, if any
```

b) The return action tests various bits in the key and performs actions as described below.

1) **Interrupt bit**

   *If set*

2) The return action tests various bits in the key and performs actions as described below. In what follows, the top of stack entry is the one immediately below the stack entry for the subprocess executing the return operation.

1) **Interrupt bit**

   If set, a scan is started down the tree towards the root from the beginning, with the subprocess at top of stack, looking for a subprocess
with interrupts armed, when one is found, the interrupt

waits if its interrupt waiting bit is on, no further action

for this interrupt. If off, it's interrupt waiting

bit is turned on and 003623 the interrupt

datum is placed on the subprocess interrupt

datum.

ii) return with process bit.

If on, the environment is restored to that of top of

stack, and appropriate parameters return action

takes place.

iii) Return bit.

If on, the return count is incremented by 1.

iv) Repeat bit.

If on, the pointer modifier is set to, "about to execute

an instruction at address PC".

v) Complete bit.

If on, the pointer modifier is set to, "almost finished with

an instruction at address PC", (note that this

bit overrides the repeat bit.)
vi) **error bit**

If only a scan is started down the tree towards the root beginning with the subprocess at top of stack, looking for a subprocess with the bit on in its error selection mask corresponding to the given error class, when one is found, the bit is turned off in its error selection mask and a new top of stack entry is made by pushing down the old top of stack entry. The new top of stack entry represents a call on the subprocess just located, i.e., its p-counter is set to the entry point of the found subprocess. Its p-counter is set to zero, its E-return count is set to 0, its class code is set to that of the found subprocess. Its top of path class code is set in the usual manner. The environment is now set to that of the new top of stack and the error class and number are placed in the appropriate locations.

Notice that the error bit is lost one scan down since it may create a new statement. This could be avoided if the error info could be held in the subprocess descriptor (as for interrupts, or held in a new entry itself.)
6) Finally next the return operation begins a scan downwards towards the root beginning with:

   The root

6) Now the return action starts a scan down the tree towards the root beginning with:

   The root

C) Now the return action examines the top of stack entry. (This is a new one if the error bit was on). It branches on the value of the pc at 0.

   i) "about to execute an instruction at the address pc"

   A scan is started downwards through the tree towards the root beginning with the top of stack. In the top of stack, the subprocess looking for a subprocess with interrupt inhibit bit on. If one is found and the subprocess is not the top of stack, the subprocess to exit the top of stack subprocess and its interrupt inhibit bit is off. Then the following takes place:

   Interrupt waiting bit is turned off. Interrupt inhibited is turned on. A new top of stack entry is made as is the error bit case. The environment is set to point to the new top of stack. The interrupt action is copied from the subprocess descriptor to the appropriate place in core. Finally we go to C) (hence C) if ) [pc = interrupt - 8]
ii) "immediate of a complicated xj instruction at address pc."

Set the environment to the top of stack, from the xj instruction locate an appropriate operation.
Now set the operation has sufficient depth to handle the F-return condition (0 is for stopper, etc.)
If so, when new stack entry as in error case,
(set pc to entry point), goto (hence c) (i)
If not, set pc = pc + 1, set pm to "about to execute an instruction at address pc", goto c
(hence c) (i) .

iii) "almost finished with a complicated xj instruction at address pc."

Set the environment to the top of stack, from the xj instruction compute an appropriate pointer offset.
If this is a normal case set pc to the new line, set pm
 to "about to execute an instruction at pc" and goto (c)
(hence c) (i) .
If this is not within range, generate an appropriate error, and an error status in error case. Then goto c)
(again, hence c) (i) .
our existing return instructions act same as his one
but w.r.t. subset of the specified params and
w.r.t. fixed key. (would firma unused params?)

4) ordinary return
   only bit on is complete bit. no params

b) F-return
   only bit on is F-return. no params

c) error-return
   only bit on is error-return. error class and error number
   are only params

d) return with params.
   complete bit and params bits are only bit on.
   brain data and brain capability are only params.

e) special return
   repeat bit only bit on. no params

f) reject on a call; want to make new function entry set pc = next sq
   pcm = "go to execute or instruction at pc" now proceeds
   to (ii) or (i) (hence iii) (i)
An external interrupt arrives with a data and a class code.

1) The given sub process is examined. If interrupt waiting is on, nothing else. Else the data is stored in the sub process and interrupt waiting is turned on.

2) The given sub process is examined. If interrupt not armed, then nothing, else as in 1).

3) Decide a foreseen timeout. The work is started with the given sub process. If a sub process is found with interrupt armed then proceed as in 1), otherwise?

B) Now examine the process

1) How on an event channel just got received an event, change PCM in top stack entry to "about to execute an instruction at PC", set a signal in the process descriptor to unhook from the event channels, reschedule the process.
ii) Hung on event channels, has received an event.
    (cause program top stack entry to be almost finished
    with complicated step instruction at ptr. Could have
    been done by the event itself?)

iii) swapped out [by quantum overflow?]

      dauntiny

   c) one for Ron Young

     when every process is swapped in after removal from
     any event channels, a tree scan towards the root
     starting with the top of stack subprocess is done, looking
     for a subprocess with interrupt waiting on, (0)
     If one is found, a check is made to see if it was interrupt
     inhibited or if it is not top of stack. If so, it is called
     as in (c) i) of (III).
a number of unclear things

b) strace "foo full"

c) what if 2 subprocess found with interrupt writing? one has interrupts inhibited?

etc.

hmm. If 1st one sets called, as command ran, sure not would find 2nd. So if 1st inhibited
I think the command should continue and find 2nd?

Also, they would take if the Guy didn't have interrupt writing even if the 2nd inhibited.

c) what if no subprocess found with interrupt enabled?

d) no subprocess found with error class EIPON in its

work.

vIII) need

set interrupt inhibit
clear interrupt inhibit
set interrupt around
clear interrupt around

This description can probably be somewhat better structured especially we make new stack entry item