Nov 30
1970

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 (FRTN)

class code for subprocess at top of stack

Top of ~~stack~~ party class code

IP list pointer

interrupt inhibited

This proposal only concerns the first 3.

~~DOC~~

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 XF instruction at address pc

iii) almost finished with a complicated XF instruction at address pc

C) The F-return count contains a non negative integer

II) variables maintained in a sub process descriptor

BIT : interrupts armed

BIT : interrupts inhibited —— instack

BIT : interrupt waiting

D : interrupt datum

D : error selection mask

III) Fancy return instruction

A) parameters as follows:

PL0 ~~key~~	0:	key
P2	BO:	data params being returned, if any
P3	BC:	capability params being returned, if any
P4	0:	error class, if any
P5	0:	error number, if any
P6	0:	interrupt datum, if any

~~B) the return action tests various bits in the key and performs actions as described below~~

~~i) interrupt bit~~

~~if on~~ 1

B) 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.

~~a) if current SP of a process has interrupt waiting (not inhibited?) process the interrupt (set up current stack SP about to execute its entry point) return + go to 6~~

i) interrupt bit.

if on, a scan is started down the tree towards the root beginning with the subprocess at top of stack, looking for a subprocess

with interrupts armed, when one is found, ~~its interrupt waiting~~ if it's interrupt waiting bit is on, no further action for this interrupt. If off, it's interrupt waiting bit is turned on and ~~the process~~ The interrupt datum is placed in the subprocess interrupt datum.

ii) <u>return with params bit.</u>

if on, the environment is restored to that of top of stack, and appropriate parameter return action takes place. including making a new TOS in case of error

iii) <u>F return bit.</u>

if on, the F-return count is incremented by 1.

iv) <u>repeat bit.</u>

if on, the pointer modifier is set to, "about to execute an instruction at address PC".

v) <u>complcat bit.</u>

if on, the pointer modifier is set to, "almost finished with a complicated ~~next~~ instruction at address PC". (note that this bit overrides the repeat bit)

vi) error bit

if on, 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 it's error selection mask and a new top of stack entry is made, pushing down the old top of stack entry. The new top of stack entry represents a call on the subprocess just located. i.e. it's p-counter is set to the entry point of the ~~sub~~ found subprocess. (-8?) PCM is set to "about to execute an instruction at address pc. Its F-return count is set to 0. It's class code is set to that of the found subprocess. It's 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 and since it may create a new stack entry. This could be avoided if the error info would be held in the subprocess descriptor as for interrupts, or held in the stack entry itself.]

~~c) Finally Next the return operation begins a scan towards the root~~

~~c) Now the return action starts a scan down the tree towards the root beginning with~~

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 pcm bit.

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

a scan is started ~~down the tree~~ down the tree towards the root beginning with the top of stack subprocess looking for a subprocess with interrupt waiting bit on. If one is found, and the found subprocess is not the top of stack subprocess, or it is the top of stack subprocess and its interrupt inhibit bit is off, then the following takes place:

otherwise, the environment of the top of stack is reestablished & execution resumes at PC.

Interrupt waiting bit is turned off. Interrupt inhibited is turned on. A new top of stack entry is made as in the error bit case. The environment is set to that of the new top of stack. The interrupt datum is copied from the subprocess descriptor to the appropriate place in core. Finally we go to c) (hence c) i)) [pc = entry point - 5]

ii) "in middle of a complicated XT instruction at address pc"

Set the environment to the top of stack. from the XT instruction locate the appropriate operation.

now see if the operation has sufficient depth to handle the F-return count in stack (0 is for 1st order, etc)

if so, make a new stack entry as in error b case, (set pc to entry point). so to c) (here c) i))

if not, set pc = pc+1, set psw to "about to execute an instruction at address pc", so to c) (here c) i)).

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

system return → Set the environment to the top of stack. from the XT instruction compute the appropriate p-counter offset.

If this is within range set pc to the new value, set psw to "about to execute an instruction at pc" and so to c) (here c) i))

If this is not within range, generate the appropriate error, and do error conversion in the error b case. Then so to c) (again, here c) i))

IV) our existing return instructions act same as in one bit with a subset of the specified params and with a fixed key. (would fix the unused params?)

A) ordinary return

only bit on is complete b.t. 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 bit's are only bits on.

Block data and block capability are only params.

G) special return

repeat bit only bit on. No params

i) ~~Recall~~ on a call, want to make a new stack entry, set pc = entry pt, pcm = "about to execute an instruction at pc" Run procedure in ~~(II)~~ III) C) D (hence III) C) i))

~~(6)~~ VI external interrupt.

A) an external interrupt arrives with a datum and a class code.
3 purposes

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

ii) The given sub process is examined. If interrupt not armed, Then nothing, else as in i)

iii) ~~A tree scan~~ a tree scan commences the root is started with the given sub process. If a sub process is found with interrupt armed then proceed as in i). otherwise?

B) Now examine the process

i) Hung on an event channel (s), not yet received an event, change pcm in top stack entry to "about to execute an instruction at pc", Extra signal in the process descriptor to unhang from the event channels, reschedule the process.

avoid swapin when interrupt has it priority ???

prevent events from arriving.

ii) Hung on event channels, has received an event.
change PCB, in top stack entry to "almost finished"
with a complicated XJ instruction at PC". [could have
been done by the event itself?] should be?

iii) swapped out [by quantum overflow?]
do nothing

c) one further thing

when every process is swapping in, after removal from
any event channels, active scan towards the root
starting with top of stack subprocess is done, looking
for a subprocess with interrupt waiting on ~~[illegible]~~
If one is found, A checks is made to see if it has interrupt
inhibit off or if it is not top of stack. If so, it is called
as in c) i) of III),

VII) a number of unclear things

a) stack "too full"

b) what if 2 subprocesses found with interrupt waiting? one has interrupts inhibited?
etc.
hmm. If 1st one sets called, as command Rcvr ensure that would find 2nd. So if 1st inhibited I think Thescan should continue and find 2nd.?
Also, Thy would take if the top guy ~~[illegible]~~ d. dont have interrupt waiting, even if the 2nd inhibition.

c) what if no subprocess found with interrupt armed, enabled, not inhibited & waiting?

d) no subprocess found with error class bit on in it's mask?

VIII) need

Set interrupt inhibit
Clear interrupt inhibit.
Set interrupt armed
Clear interrupt armed

IX) This description can probably be somewhat better subroutinized esp. the make new stack entry items