Speed phreaques, scheduling, ~~tt~~ compactification, ECS code, DAE, + other theological questions made simple.

SF's are supposed to give damn fast responses, like for a real-time somethingoranother.

Thus, when an SF is {awakened / interrupted}, the scheduler has to do something snappy. The current guy has to be suspended + the SF fired up within some short time $\leq l_1$. Normally, system code being reentrant + all, enuf of the CM to accomodate the SF could be copied by brute force to ECS + the SF brought in + run. The CM is then restored + allowed to run. I know of some problems:

1) The CM may be in the middle of a system call which is executing ECS code in some buffer. The SF may wipe out the buffer. It seems like the contents of the buffers have somehow to be preserved.

2) ECS may be all bent 'cause compactification is in progress. The compactifier has to be told to cool it. ~~+ allowed~~ It will require some piece of time l_2 to get itself straight. In this sense, it must

be incremental.

3) The allocator isn't really reentrant.

The fight between l_1 + l_2 is obvious. ~~Two~~ Some things should be noted:

1) ~~If the~~ The SF's ~~may~~ may have to be recompiled.

2) The compactifier may be moving something gargantuan. Either

a) it is allowed to finish

or

b) a mechanism for half-moving something has to be thunk up.

3) The SF better hadn't cause anything to be allocated (even destroyed is annoying).

4) If there's more than 1 SF, things get complicated fast

5) How long can I-LOCK remain set?

Details of initiating a SF

A process may be fired up by:

1) getting an event
2) receiving an interrupt
3) being created.

For now, we consider only 1 (& will include 2 & 3 later if it falls out (or if we're forced to)).

A) The interrupt code calls the event code calls the scheduler. The scheduler detects that it's a SF awakening.

B) The scheduler may have to determine what was interrupted:

1) user
2) system

so as to get into (& out of) monitor mode correctly