speed phrenes, scheduling, & compactification.

ECS code, DAE, & other theological questions.

SF's are supposed to give damn fast response,
like for a real-time something-something.
Thus, when an SF is {awakened / interrupted}, the scheduler
has to do something snappy. The current
guy has to be suspended & the SF fired
within some about time T. Normally, being running
enough of the CPU to accommodate, the SF
could be copied by write force to ECS
& the SF brought in & run. The CPU
then restored & allowed to run. I know
of some problems:

1) The CPU 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 come
like the contents of the buffers have
somehow to be preserved.

2) ECS may be all hale 'cause
compactification is in progress. The
compactifier has to be told to cool
it down & allow some piece of time E2 to
get itself straight. In that sense, it must
be incremental.
3) The allocator isn't really efficient.
   The fight between $h_1$ and $h_2$ is obvious. Two some
   things should be noted:

1) The SF's map may have to
    be recompiled.

2) The compactifier may be moving
    something gadgetmanian. Either
    a) it is allocated to finish
    b) a mechanism for half-moving
       something had to be thought 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 (we'll include 2 + 3 later if it fails out of the queue or fails 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 (or out of) monitor mode correctly.