Tasks:

1. Add SCHED\_SHORT, OVERDUE\_SCHED\_SHORT statuses (maxim)
2. Add (define) sched\_setschedular() system call with all the wrappers (dean)
   1. **Steps for changes in the sched\_setscheduler system call(sched.c):**
      1. check the parameters are valid
         1. line 1169 in sched.c should add a condition for policy!=SCHED\_SHORT
         2. line 1179- we should define in the sched\_param sched\_priority for the short proccess and add there a condition accordingly
      2. 2. Extract the given process from it's current relevant list (e.g from the OTHER list)
      3. Update relevant fields for the SHORT policy
      4. Add the process you extracted to the relevant SHORT list
      5. Turn on the context switch flag
3. Add request time & number of processes properties somewhere to all the SCHED\_SHORT processes. (Arye)
4. Go over the schedule() function before the context switch (ALL)
5. Implement our functionality in schedule().

As far as I understand, scheduling event is calling this function. So all we need to do is to check if there are RT processes:

if ( sched\_find\_first\_bit(array->bitmap) < 100 )

if so, continue as usual. Other-wise, choose short process the same way schedule() chooses process, but from the “short” prio\_array\_t instead of from “active”.

If there are no short processes, try to find OTHER\_PROCESS ( 99 < priority < 140 ). When there are no processes at all in the active array, switch between expired & active (already implemented, just make sure that’s what happens before next step).

If all these processes are not to be found, choose a process from the overdue\_queue. This should probably be implemented just before the part the process switches to the swapper.

1. Figure out where the hell is the runqueue instantiated, & allocate our fields (short & overdue\_queue. BTW maybe that’s unnecessary, hopefully that’s the case, because a hint in the h.w. description implied this assignment can be implemented without kmalloc. Maybe the aux functions that manipulate prio\_array\_t & list\_t do it for us?)
2. Implement changes in sched\_setschedular(). Probably add all the necessary changes in case the user wants to make a short process. (add it to the “short” prio\_array\_t & NOT to “active”)
3. Implement changes in do\_fork(). If the father is a short process, we should update the new processes fields to make it a short process with half the time & half the trials. (Q: *should we do something different for short\_overdue processes?*)
4. Implement changes in do\_exit(). Remove the short processes & short\_overdue processes from their location & not look for them in expired/active.
5. Implement changes in scheduler\_tick()

We need to figure out whether we should change something in it. I think that maybe in order to make a short process leave the cpu when RT process arrives, some unique implementation maybe required (something with this need resched switch)

IDEA:

Since the short-processes behave as if they are between RT processes & OTHER processes, but use the same priority range as OTHER processes, perhaps a good implementation would be as such:

We will add another prio\_array to the runqueue. It will hold the SHORT processes.

We will also add another queue that will store all the SHORT\_OVERDUE processes.

We will go through all the scenarios in which a process can be switched, and apply the changes according to the h.w. description - “short” after “RT”, “short-overdue” after “other”.

Maxim’s addition: perhaps if we already create another prio\_array\_t, we can move RT processes to it, thus making the choosing of a process easier.