MP2

Group Member:

Dashuai Qin(dqin3), Edgar Cortez (ecortez2)

Implementation and Design Decision:

Task_Structure:

We used a Linux list_head linked list to store PCB structure for all the tasks, we stored the process id, period information, computation time, along with the state of the task and the wake-up timer it uses.

Wakeup_Timer:

The wakeup_timer should wake up the kernel thread when the next period is going to start. It sets the process it belongs to state to “READY” and proceeds to wake up scheduler thread. We made sure to save the state using spin locks since inside the interrupt time is very important and in case of more than one timer waking up.

MP2_Write:

Parse all the arguments from userapp function, and goes to different functions depending on the commands, “register, yield, deregister”.

Registration:

We first initialized a new task structure using slab-cache since it would be best efficient. After getting all the arguments from userspace, we decide to store them in our task structure and check if the task passes admission control, which is the UB test. If it passes the test, we register the task by putting it into the linked list and sort it according to its pid. In this way, it would be much more convenient later when we want to find the highest-priority task. If it did not pass the test, we do nothing and free the structure. Because we cannot use floating point since it is expensive, we converted the ratio into integers. We made sure to place appropriate locks for our data structure for race conditions as well.

Yield:

In the yield function, we first check if the time has passed the deadline and then put the task into sleep, and start the timer to wake it up at the next period. It also wakes up the kernel thread so it can start scheduling.

De-Registration:

In the deregistration function, we list through the list to free all the tasks in the list. We also delete the timer associated with it.

Thread_Dispatch:

After initializing the kernel thread, it will trigger context switch when it gets woken up. In our do_work function, we checked if the current task will get preempted or not by looping over all our tasks and checking if one is ready and has a higher priority (lower period) than current one. Then we schedule the highest-priority task that is ready.

Test Application:

The main function takes in three arguments, period, computation time and job numbers. In the main function, register got called first. Then we read the proc file to see if it got registered. If it does, we call yield to signal scheduler that the task is ready. Then it goes into a loop until it has been scheduled for number of times equal to job numbers. It prints out the wake up time and process time in the loop.

Test Method:

You can run our user app with ./userapp [period] [computation time] [num jobs]
We ran two instances using $./userapp 5000 4000 2 & ./userapp 12000 4000 5 &