Group 17
Ahmed Shabban - shaaban2
Caleb Perkinson - perknsn2
Vibhakar Vemulapati - vemulpt2
Xiejia Zhang - xzhan121

Kernel Thread:
This thread serves as the dispatcher that performs all the context switches. It is responsible for finding the task with READY state that has the smallest period and switching to it. Sleeps most of the time, is awoken by either a YIELD or a timer expiration. Once it wakes up, it initializes 2 pointers to mp2 task structures, one for the next task (i.e. task with lowest period and READY state), and one to the currently running state. 2 helper functions are used to set these pointers. If there are no tasks in the READY state the _getNextTask helper function returns a null pointer. If there is no currently running task the _getCurrentTask helper returns a null pointer as well. In the dispatcher we check both of the pointers for null. If the next task pointer is not null, then we set its priority to 99 and instruct it to wake up. We also set its status to running. If the current task pointer is not null then we set the currently running tasks priority to 0 and set its status to READY. Since our helper function _getCurrentTask only returns RUNNING tasks, the case where the task has yielded is handled. After performing the previous operations the dispatcher thread goes to sleep.

Timer Handler:
This thread will reset the status of a task to READY at the beginning of the next period. A pointer to the mp2 task structure the timer is associated with is passed in as the argument to the timer. All the handler does is set that tasks status to running and wake up the dispatcher.

Registration:
To register with the Real-time scheduler, a user process writes to the /proc/mp2/status using the following string, “R, PID, Period, Computation”, where PID is the pid of the process, period is the time given per loop, and computation is the time required per period to do actual processing. When the user registers a new PCB (process control block) is created and stores vital data about this process. The user must then call YIELD, which is discussed below.

Yield:
To yield the running job a user writes to /proc/mp2/status as usual with the identifier ‘Y’ and their pid delimited by a comma. The user will yield when the work they needed to

do for that period was completed.  This is handled by the kernel by putting the formerly running process to sleep and configured its wakeup timer to fire at the start of the next period.  When this timer fires, its handler will wake up the kernel thread to schedule new work to be done.

De-registration:
To de-register with the Real-time scheduler, a user process writes to the /proc/mp2/status using the following string, “D, PID”, where PID is the pid of the process. The de-registration handler then safely loops through the list of mp2 task structures until it finds the task and removes it from the list. It also cleans up the timer and all allocated memory for the task.

Admission Control:
When a new process tries to register with the real-time scheduler, the kernel will first check whether or not the process is capable of being scheduled.  This is determined by doing the utilization bound test which says the sum of the ratios of the process’ computation time to its period must be less than 0.693.  The admission control will check this criterion with the new process and if it still agrees then the process will be scheduled, otherwise it will not schedule.

**How to run:**
Run make, insert the kernel module( mp2.ko). Then run userapp. If it is run with no arguments, it uses the default period and number of jobs. You can pass in 2 arguments (period, number of jobs), to change the coresponding values.