**Programming Assignment 5**

# Objective

The objective of this assignment is to use POSIX-based semaphores to implement a simulated real-time scheduling algorithm.

# Assignment: Implementing a pseudo real-time scheduler

The idea is to write a C program that simulates a Rate Monotonic (RM) real-scheduler. The scheduler will create *n* number of threads (based upon the tasks defined in the task set file). Then simulate the scheduling of those threads using posix based semaphores. Each thread will wait in a while loop waiting to acquire the semaphore. Once acquired the thread will print-out just its task name then go wait for the next semaphore post by the scheduler (Only one function should be needed to implement the thread tasks). The scheduler will release or post a semaphore (so *n* tasks means *n* sempahores) based upon the RM scheduling algorithm. A “clock tick” will be simulated through each iteration of the main scheduling loop (i.e. one iteration first clock tick, two iterations, second clock tick,). Assume all task are periodic and released at time 0.

**Scheduler Program Implementation**

The RM scheduler program (*rmsched.c*) that takes three arguments from the command line (no prompting the user from within the program).

1. To start the *rmsched* program

*./****rmsched*** *<nperiods> <task set> <schedule>* where <nperiods> defines the number of hyperperiods, <task set> is a file containing the task set descriptions, and scheduler is a file that contains the actual schedule. Hyperperiod is LCM of periods

The format of the <task set> file is as follows: (hyperperiod = 24)

*T1 2 6*

*T2 3 12*

*T3 6 24*

Where the first column represents the task name, the second column represents the task WCET and the third column represents the task period.

The example format of the <schedule> file is as follows:

*0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15*

*T1 T1 T2 T2 T2 T3 T1 T1 T3 T3 T3 T3 T1 T1 T2 T2*

Where the first row represents the time (only needs to display once at the top of the file) and the second row represents the actual schedule for one hyperperiod. The next hyperperiod should begin on the next row.

1. The main process is to create the *n* threads and the *n* semaphores then determine which thread to release based upon the RM scheduling algorithm. The main process should also check to ensure that the task set is actually schedulable before attempting to create a schedule.
2. The task thread only has to wait for the appropriate semaphore then print-out its name to the <schedule> file and wait for the next semaphore post.

**Error Handling**

Perform the necessary error checking to ensure the correct number of command-line parameters as well as checking for capability to open/create/read/write to the two files. Make the sure the task-set is schedulable if not just abort the program.

# Grading

The program will be graded on the basic functionality, error handling and how well the implementation description was followed. Be sure to name your program **rmsched.c** (no extra characters, capitals) Note that documentation and style are worth 10% of the assignment's grade!

# Submission

The program source code along with sample output should be posted onto Blackboard by the due date.

Thread Function:

While (1) {

        sem\_wait(semid)

        printf("Task Name: %s is running \n");

fflush(stdout);

}

Print idle if nothing to run

Cast semid to void \*

Choose which to run if tie in priority

Scheduler:

for (nperiods < hyperperiod) {

determine which task gets to run

release task semaphore

}