CompE571 Homework 3 Report

**Software:** Visual Studio 2015

**Language:** C#

General Instructions: To run our project, download and extract the provided project archive. To run from the command prompt, please navigate to the directory where the executable file for the project is located (can use cd command for navigation). The form for executing the project is “HW3.exe input1.txt EDF null”.

**Description:**

*Basic EDF* – We chose to use C# for the ease of creating a modifiable task list. First, we created a class called task which would hold all of the tasks included in the input file(s). The task class includes assignments for task name, deadline, and WCET for each CPU frequency. Also included in the class are static variables which are used as placeholder/restoration instances for WCET and deadline. After reading and parsing the input file by line into a dynamically allocated string variable (using spaces or new lines as delimiters), we load the variables such as number of tasks and total execution time into variables for use in the program. We use a try, catch block so that the program can be run in Visual Studio without any command inputs. If the executable file is called with no inputs, a message will be displayed showing that the file cannot be read. The rest of the strings are loaded by index into their corresponding task name (i.e. w1, w2, etc.). Then, we create a list of objects which holds the full task set of tasks to be operated on.

Another reason for using C# is the ease of using a foreach loop. However, for our main looping scheme we use a normal for loop which measures current time in the system and ends at the total execution time. Our two foreach loop system comprises of one checking relative deadlines of each task compared with the relative deadlines of the other tasks in the system at any given time, while the other handles execution time, frequency, idle, and output sections. The deadline scheme makes sure that the deadline for a task is the least for all tasks in the system at a current time, and then subsequently schedules this task for execution by setting it to the scheduled task.

In the second foreach loop, the WCET, in this case at the highest frequency state of 1188Hz, is decremented for the scheduled task at each iteration. Then we check to see if all tasks in the system have completed their execution during their current period, and if so, we schedule an idle period for the scheduled task, while also changing the consumed power to the idle power state. If the WCET for the 1188Hz variable is zero, we can update the deadline for that task and reset its WCET variable. The last if statement in this section produces the output based on a context switch in the system; because EDF is preemptive, we must output the last scheduled task whenever there is a context switch. Start time of a task is computed by setting this value at the completion of the previous task, end time is computed by current time in the system minus this start time. Lastly, we output the result of the last scheduled task using a console write. In the case that total execution time is reached, we must also output the current scheduled task in the system, regardless of whether a context switch has occurred.

Energy consumption, percentage of time spent idle, and total execution time are all computed and output to the console at the end of the scheduler. We use a filestream and streamwriter to take the contents of the console writes during execution and send them to a new text file, named depending on the scheduler type, which is created in the same location as the executable file for the project. For all intents and purposes, we believe our scheduler for EDF is feasible and realizes all deadlines in the system.

*Full disclosure:* When calling our program from the command prompt and selecting EDF or RM, please input “null” as the last argument after selecting EDF or RM. We were getting errors when leaving this blank and passing the args() variable a null value. This doesn’t affect execution of the program in any way.

*Basic RM* – The basic RM algorithm was not much different than our EDF scheduler. Starting off, we copied and pasted our EDF scheduler and tried to figure out the main difference. Obviously, RM must schedule not based on relative deadline, but on straight deadline relative to the current time in the system. The main difference with RM is that we do not update the deadline for the task after its completion. For example, if a task has a deadline of 200 and meets it’s deadline before that mark, it’s static deadline changes to make sure that it doesn’t execute again within the current period, but it’s deadline is unchanged so that at every 200 period, this task must execute no matter what if it’s deadline is the shortest compared to all tasks. The scheduler is unchanged and picks the task with the shortest relative deadline compared to all other tasks and using the current time in the system.

Again, we believe our RM scheduler works as intended and outputs to a text file showing all scheduled tasks, their consumed power, and the system statistics which describe the execution.

*EDF EE* –

*RM EE* –