Computer Engineering 571: Embedded Operating Systems

Programming Assignment 3: Dynamic Variable Frequency Scheduling Report

Skye Russ (132722795)

Introduction:

Dynamic frequency scheduling uses varying CPU frequencies to achieve lower total power for a set of tasks. It requires all tasks to be known before the scheduler is called to then create a static schedule for periodic tasks. Some considerations that need to be made are that all tasks must still meet their deadlines and the slowdown of CPU frequency must decrease the total power used by any given task instance.

This idea is then combined with the Rate Monotonic (highest priority task is the one with the least computation remaining) and Earliest Deadline First (highest priority task is the one with the soonest deadline) scheduling algorithms to create energy efficient versions of each algorithm.

Implementation:

The provided script "dvfs.c" implements Rate Monotonic (RM) and Earliest Deadline First (EDF) scheduling with additional functionality to find the most energy efficient CPU frequency for the task set. At execution time, a file is provided that includes the number of tasks, number of time steps to schedule, energy consumed at given CPU frequencies (1888Mhz, 918Mhz, 648Mhz, 384Mhz, Idle) in milliwatts, and then each task is described by name, period, and the worst-case execution time at the four CPU frequencies. Once this file is parsed, tasks are stored in a Task Description struct for easy use. The user passes which scheduling algorithm desired as well as if the schedule should be optimized for energy efficiency.

Each scheduling algorithm is defined in its own function with one function defined for energy efficiency. The energy efficient function calls a recursive helper function that iterates through all frequencies for all tasks that provide a decrease in power consumed when compared to maximum CPU frequency. For the example input (detailed in Figure 1), 432 combinations of CPU frequencies across the five tasks were tested. The lowest energy is stored and then the script prints the results of the tasks at the given frequency.

The definition of the RM function and EDF function are identical except for the priority definition algorithm which is used by the scheduler. The function first computes the total CPU utilization for the task set, if this utilization is above 1, the function immediately returns -1. The current active task is stored as an index value on an array of remaining execution time for each task. The function then iterates through timesteps starting at 0 and ending at the specified limit. The loop first checks for any tasks that have missed their

deadline or are going to be restarted by taking the modulus of the period for each task compared to 0 (if this value is true, the scheduler also needs to be called so a flag is set because a new task has arrived). If any tasks missed their deadline, the function immediately returns -1 indicating a failed schedule. If no tasks have missed their deadline, the function checks if the current active task has no remaining computation which would also set the scheduling flag to true. If the scheduler is called, it then compares the available tasks (indicated by those tasks having remaining computation required) using either Rate Monotonic or Earliest Deadline First priorities. If no task is selected, the current active task is set to -1 indicating the CPU is idle. After the task is selected, the loop decrements that task's remaining computation by 1 and continues. Once any task finishes its computation or the scheduler is called, the function prints the start time, task name, CPU frequency, end time and power consumed (in Joules) for the given task period.

There is an additional flag for the function that suppresses print statements which allows for efficient repetitive calls to the scheduling functions (which are used by the energy efficient option.

Results:

Below is an example task definition as well as the outputs of the "dvfs.o" script called using RM, Energy Efficient RM, EDF, and Energy Efficient EDF. Given the following task sets (figures 1 and 2), the schedules for Rate Monotonic (maximum CPU frequency), Earliest Deadline First (maximum CPU frequency) and Energy Efficient Rate Monotonic (minimum CPU frequency) and Energy Efficient Earliest Deadline First (minimum CPU frequency) were computed by the script. Each successful schedule then returned the total energy consumed and percentage time the CPU was left idle.

Figure 1: Task Set 1 and CPU Frequency Costs (mW)

```
5 1000 625 447 307 212 84
w1 520 53 66 89 141
w2 220 40 50 67 114
w3 500 104 134 184 313
w4 200 57 74 103 175
w5 300 35 45 62 104
```

Figure 2: Task Set 2 and CPU Frequency Costs (mW)

```
5 1000 625 447 307 212 84
w1 520 53 66 89 141
w2 320 40 50 67 114
w3 500 104 134 184 313
w4 450 57 74 103 175
w5 300 35 45 62 104
```

Table 1: Energy Consumed and Idle Time for Task Sets

Task Set		RM	RM EE	EDF	EDF EE
Input 1 (Figure 1)	Power Consumed (J)	--	--	577.933	519.09
	Idle Percentage	--	--	0.087%	0.0%
	Execution Time (s)	--	--	913	1000
Input 2 (Figure 2)	Power Consumed (J)	546.014	412.135	546.014	374.952
	Idle Percentage	0.146%	0.161%	0.146%	0.0%
	Execution Time (s)	854	839	854	1000

For task set 1, there is no valid Rate Monotonic Schedule. This is shown by process "w1" failing to meet its deadline at time = 520. The energy efficient schedule shows the same result. However, Earliest Deadline First creates a viable schedule with 0.087% idle time at maximum CPU frequency and 0% idle time at reduced CPU frequencies. The energy efficient schedule saves 58.84 Joules of energy.

For task set 2, interestingly the Rate Monotonic and Earliest Deadline First algorithms output the same exact schedule. This results in identical execution time, idle time and energy consumption. However, the energy efficient earliest deadline first algorithm saves 171.062 Joules whereas the energy efficient rate monotonic algorithm only saves 133.879 Joules. This shows the benefit of optimizing out all the idle time and reducing the CPU frequency of every task as much as possible.