CSC 460

Design and Analysis of Real-time Systems

Project 2

March 6, 2020

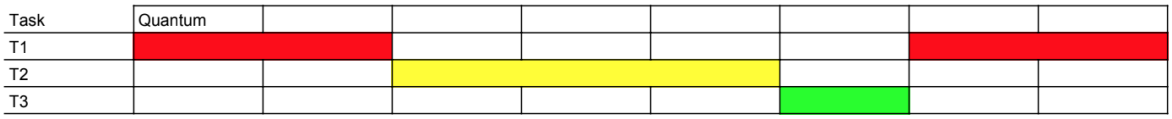
Joel Kerfoot – V00855134

Braiden Cutforth – V00853754

Introduction

There are many different scheduling algorithms designed for computer systems. Earliest deadline first, rate monotonic, round robin and weighted round robin to name a few. This project focuses on implementing weighted round robin in a real time operating system on an ATMEGA2560 chip. Weighted round robin is a scheduling algorithm where each task is run in a random order. The length of time a task is run is based on the weight assigned to the task. See figure 1.

Assume that T1 runs before T2, and T2 runs before T3. The order is arbitrary



T1, W = 2
T2, W = 3
T3, W = 1

The scheduling is cyclic, so after T3 is run the cycle repeats. As we can see, the weight determines the number of time units (quantums) the task runs.

Implementation

Project 2 was done with the help of an existing implementation of round robin scheduling. The code was modified in order to accommodate weighted round robin scheduling. There were two main changes that were needed first, the ability to create tasks with weights second, only context switching once the task has run proportionally to its weight.

To create weighted tasks `Task_Create_WRR` was added which takes weight as one of its parameters. When the kernel is active the weight can be set directly. If the kernel is inactive `Kernel_Create_Task` is called. Weight was added as another parameter to this function so that weighted tasks can be accommodated. For other tasks a weight of 0 was used. Ultimately, `Kernel_Create_Task_At` is called which also needed the new weight parameter and handles setting the weight for weighted round robin tasks.

All changes needed for context switching were done in `Task_Next_2`. When this function is called, and the current tasks has priority 'RR' the 'executed_ticks' of the task is increased by 1. If the executed ticks are less than the weight of the task the function returns, and the task continues to run. Once the executed ticks equal the weight of the task the context switch is done so that the other tasks can use their share of the cpu.

Testing and Profiling

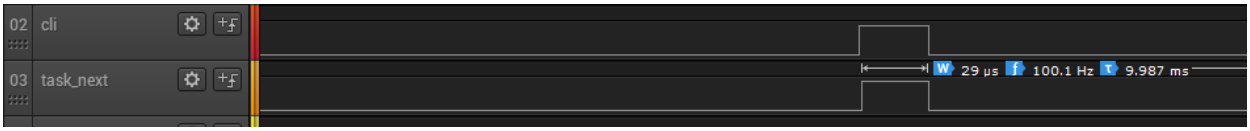
Testing and profiling were done using the logic analyzer. Figure 2 shows the tasks running on the system. Each task will raise a specific pin. Each task's pin was connected to its respective channel on the logic analyzer (ie. Task 1 is on Channel 1). Channel 1, 2 and 3 were profiling tasks called using the weighted round robin algorithm, using weights 1, 4 and 16 respectively. As we can see, channel 1 is

executed for 4 times less time than channel 2, and channel 2 is given 4 times less time than channel 3. Channel 4 was called using the round robin algorithm, thus defaulting to the lowest weight.

Figure 2 – Task profiling

Profiling for the time to context switch and interrupt disabled time is very similar to what was done in exercise 10. Getting the Task_Next() function to set a pin high and low during the context switch allows us to time the context switch. Getting the disable_interrupt() and enable_interrupt() to set a pin high and low respectively allows us to time how long the interrupts are disabled. These timings can be seen in Figure 3. Channel 02 shows the interrupt disable time, and channel 03 shows the context switching time.

Figure 3 – Interrupt disable time and Context Switch Time



The interrupt is disabled for 31 microseconds for a context switch. The context switch takes 29 microseconds.

Conclusion

In this project, weighted round robin scheduling was implemented on the ATMEGA2560 chip. Weighted round robin works by allowing tasks with higher weights to run for longer periods of time. Tasks are run in an arbitrary but consistent order, i.e. once an order is established it is maintained.

The implementation was mostly based on the provided sample code. A few methods were added for handling weighted round robin tasks. Additionally, some other functionality was added to assure the tasks got the proper amount of runtime.

The scheduling was tested with 4 tasks, where tasks 1, 2, and 3 were scheduled using weighted round robin and task 4 was scheduled using round robin. The time taken for a context switch was 29 microseconds and the interrupt disable time was 31 microseconds.