CS 3410 / ECE 3140
Matthew Daniel (mrd89)
Ankush Rayabhari (ar2354)
Lab 5 Report
2019-April-25

Overview

The purpose of this lab is for us to create a system to queue real-time and non-realtime processes together and run them in order of their deadlines. By doing this, we were able to get a hands on experience with how a real operating system might handle such events as well as a better understanding of the general concept of a real-time schedule and algorithms surrounding this.

Code Overview:

- Changes to process_t
    - We added a start and deadline field to each process that will contain the appropriate realtime_t
    - These will be null for a non real time process.
- System Clock
    - To accomplish the system clock, we setup PIT1 in process start to trigger the PIT1_IRQHandler every 1 millisecond. In every execution, we increment the current time by 1 millisecond and then enable the timer to start executing again. By default process_select ignores all interrupts so while busy waiting we used the __enable_irq() and __disable_irq() functions to restart interrupts. Here, PIT0 will not trigger another context switch because it's TFLG wasn't cleared but PIT1 will. Since we defined PIT1 to have higher priority, it will interrupt the looping and keep incrementing the timer.
- Real Time Process Create
    - This is identical to the non-real time process create except we set the additional fields of start and deadline to be the realtime_t objects instead of NULL.
- Process Select
    - To handle both real time processes and non-real time processes, we kept both in a single queue that was always sorted by start + deadline so this way we choose the earliest deadline time first. Any non-real time process (i.e. a NULL deadline and start) would be put at the end of the list.
    - We implemented two helper functions that insert a process into its correct sorted position using an insertion sort algorithm and another that selects the earliest possible process. If no processes are ready at the current time, it busy waits until that time is reached and then selects the earliest process at that time.
    - To manage keeping the processes in a sorted order, we defined a custom comparison function that returns the difference of the sum of the start and deadline for each process.

Test Cases/Code Review:

For this lab, we are using four different test cases to be extra careful with our program. We identify a few worst case processes and deadlines and used those to model our test cases.

Test 0

This code was given to us to make sure the basic functionality would work and it does a pretty good job of checking a few key events. Firstly, it is a basic way to test that the real-time process gets priority over the process that is not real-time. Secondly, it makes sure we do not run the real-time process until a second has passed. In this test the non-real time process runs twice since the real time process did not meet its start time. Once the start time was reached, the processor selected the real time to run till completion followed by the remaining two blinks of the red LED. Finally since we missed the deadline of the real time process, the green LED blinks twice.

Test 1

In this test case, we have a single real-time process that has a two second start time and a deadline of 10 seconds. The purpose of this test is to make sure that the process.c can correctly handle a “busy wait” state where it loops through the process_select until there is a process ready. When we run this test, the blue LED blinks five times and is within 10 seconds. We changed the green LED to blink for as many processes were completed within the deadline. The LED blinked once at the end as expected.

Test 2

This test case is another extremely basic one that we wanted to ensure that the non-real time processes would still queue in the correct order and execute concurrently. The output was the red and blue LED’s blinking at the same time followed by a blue blink. This is because a non-real time process is placed in the back of the queue and the next is allowed to run. The green LED was programmed to blink at the end if either a real-time process was met or not met, and since neither processes are realtime, no green light came on.

Test 3

This was the most intensive test case that we did for this lab. We added four processes to the queue: three real-time processes and one non real-time. Each of the real-time processes had the same deadline of ten seconds but they had start times of 2 seconds, a millisecond, and 1 second respective to their order. We proposed that since at t=0, the only process that is allowed to start is the non real-time and thus the blue led blinked once quickly. Next, the second process (with 1ms start time) flashed the red LED 3 times, next was the second process (1s start time) ran the green LED, the first process (2s start time) runs the Blue LED, the non real time finishes its final blinking starting with red as it blinked blue once, and finally after a short delay the green LED blinks three times indicating that we met all three 10 second deadlines. This code worked exactly as planned making us fairly confident in our code.

Test 4

In this final test, we created the first process with a start time of 2 seconds and 1ms deadline and a second with the same start time but a 2ms deadline. We wanted to make sure that the processor would run the second process first since it has the earliest deadline. Secondly, we tested to make sure that the first process would not interrupt the second process once it started running. Our output was the second process, the first process, and then two blinks showing that the processes had both missed their deadlines.

Work Distribution

We started working on this lab by meeting together on Saturday, April 20th to begin the basic code. We started by reading through the lab 5 instructions and tried to make a basic plan on how to accomplish this task given to us. Using pen and paper, we had a plan set out as detailed above with the different functions we made. We used a pair programming model to accomplish the lab where Ankush was the coder and Matthew the driver. Since Ankush has a well-working VM and is faster at coding, he typed a majority of the code while Matthew helped describe what we needed to get done and the major ideas. During this time, we were able to collaborate quite well by bouncing ideas off of each other and coming to a consensus on how to proceed. This communication was very helpful for both parties and we believe we made a good team working on this lab. During this initial meetup, we were able to get a mostly working code but there were a few bugs.

The next Thursday, we met up to debug our code. This was a fairly painstaking task since we only had a few syntax/logical errors, but were able to solve these issues after a few tries and modifications. The issues we had were mostly due to incorrectly setting the priorities of each interrupt request after enabling them which according to the ARM documentation doesn't actually change the priorities. This problem manifested itself as the interrupts being disabled during process_select, but upon making the change, we started passing the given test case. Finally, we began writing test cases and tested them.

Finally, we got together on Friday to finish with the report and adding some helpful comments and general debugging. To share the code, we used a combination of email for any quick intermediate changes and Github for the proper version of the codebase.