ECE 3140 / CS 3420
Matthew Daniel (mrd89)
Xinyi Yang (xy98)
Lab #3
2019-March-14

Objective

The main purpose of this lab is to give us a better understanding of how processors run programs concurrently. This requires many steps, but specifically three programs that start, switch, and initiate the processes. This also incorporates the clock interrupts that we learned in the last lab to initiate the switching of the program.

Main Program Report

Process_Create

The overall function of process_create is to start the data structure for the program switcher. To do this we created a function enqueue(). Enqeue takes in the current process_t and with a simple if statement, adds the process to the queue as long as there is sufficient memory for it. Finally it creates a new NULL in the next place of process. We start process_create by turning off the interrupts to start process_stack_init, which is from the 3140_concur file, and then turn interrupts back on.

Once the stack has been enabled, we allocate memory using malloc() and check to make sure there was sufficient memory before continuing. Finally, using the struct syntax, we set the values for the stack pointers, size, and the next to be NULL and call our enqueue code which is explained above. This program is probably what gave us the most confusion when writing, since part of it was just understanding the code we were given in the lab.

Process_Start

To start a new process, the clocks must be enabled which was almost identical to that of the previous lab.The major difference from the previous lab to this was the amount of delay between the interrupts since we want very fast process switches. Then we called the process_begin() which initiates the first process. process_begin() on its own is extremely simple and allows for global interrupts and places the syscall into the scheduler.

Process_Select

The purpose of this section is to swap the current process for the next process that is in queue to make sure that each program gets a time to run. This occurs every time the PIT0 interrupt occurs which we set to DEFAULT_SYSTEM_CLOCK /100 which is approximately 1ms. Following an interrupt, the program checks for a few things: if the process is running, process has completed, or no currently running process. If the current process is not empty (and thus still running), the current process is placed into a temporary location and updates the stack pointer to add it to the end of the queue. If the process is empty, the program then checks if the current process and queue are empty. If this is true, then there is no program to jump to, and

thus results that the program return NULL. As long as they are not empty, the program goes into the next section which has to start by turning off interrupts. The reason for turning off interrupts is to make sure we do not get interrupted while initializing. If it were to interrupt so fast that the processor could not even make it through the process_select, there would never be much progress because the program would spend too much time switching programs. Once the interrupt is disabled, we use process_stack_free which returns a pointer at the end of the stack near the end of the allocated region. Then we turn interrupts back on, remove the current process from memory, and then change the current process if there is another process in the queue.

To finish this program, we set our selected process to become the process that is currently running. Then we run the function dequeue().In that function, as long as the queue is not empty, then it goes to the next process which removes the first process from the queue. We do one final check to make sure the current process is not NULL. If it is, then we return NULL which means no program will run, else, we will return the current process stack pointer.

Code Testing

We were given the t0, which creates two programs: p1 and p2. These programs turn on the red LED (for p1) and the blue LED (for p2) 5 times each. There are a few things that can occur when this is run. Firstly, if nothing at all happens, then we know that the process_start and/or process_create has major issues with it. The most likely scenario is that the process_select has an error which will result in either just the p1 running, or have p1 run and then p2 runs. We tested the code and the two LED's blink at the same time which means it passes this test case!

Test1: For this test, we defined p1,p2,p3,p4 which blink the red LED 5 times, blink the blue LED 5 times, blink the green LED 5 times, and blink the blue LED 2 times respectfully. The first run, we fun the first three programs which, if the program is correct, makes each color LED blink 5 times simultaneously. The second makes the blue and red LED blink simultaneously. Finally, we tested running the p4 once which makes the blue LED blink twice. We ran through these in one file to be sure that the program can run anywhere from 1 to 3 programs at once without any errors. We passed each of these test cases which gives us confidence in the code.

Test2: This code is very similar to that of test 1 but we are testing to make sure we can make each LED blink an individual number of times. We ran the red LED 2 cycles, blue LED 8 times,and the green LED 4 times. Running these at the same time gave us the following output: two blinks of all three LEDs, 2 blinks of the green and blue led, and then 4 blinks of the blue LED. This as well as test 0 and test1 passed, which makes us extremely confident in our code!

Work Distribution

For this project, we met over the weekend to discuss key parameters in the code design as are listed above. The three major components based on the lab manual are: creating, starting, and selecting the correct process while making sure it is a "fair" switch. For us, the hardest part was getting started understanding what the overall function of our code was and

how to incorporate the given programs. After reading through the given files and rereading notes from discussion, we started creating the code.

The code design was fairly straight forward with the given information in the lab manual as well as in slides from discussion. There is new syntax that Matt did not understand at first which was a little frustrating for him at first. After the initial meeting, we separated and worked on the project separately since scheduling was an issue with our schedules. We were in contact regarding the progress of each other's code and any issues we were having via text. Xinyi was able to complete the lab correctly first, and sent the code to Matt to review and look over what he had different. Next, we met on Wednesday evening to make tests and run them on the board and it appeared to work correctly! We were sure to keep brief but meaningful comments throughout the code to keep it understandable from an outside perspective!

For collaboration as laid out above, we met in the early stage after we both had a chance to read the files and manual. From there we made a basic game plan on how to complete the lab, and then worked individually for a few days. We kept in communication via text on progress updates and any struggles we had completing a task. To share the code, Xinyi uploaded the files via CMS following its completion.