Lab 2 writeup

Rayyan Khan, khan274, 2/22/2023

3.2

Setting count as a static variable allows count to maintain its value even after the program returns, allowing it to increment when the program is called again. Additionally, because divzero2 triggers an interrupt, EIP, CS, and EFLAGS are all pushed onto the stack to maintain the state of the program. This means that instead of simply popping EIP with ret, we must use iret to pop EIP, CS, and EFLAGS all. I additionally had to add 40 to ESP, but after having completed the whole lab, I realize there are more efficient ways to do it.

3.3

The asm interrupt is not a hardware interrupt (it is a software interrupt), and therefore it does not try to re-execute the process that was interrupted. A hardware interrupt will save the location of the attempted process and try to re-execute it after the interrupt is serviced, thinking that there was a problem that was resolved and now it can proceed. Therefore, the manual division by zero repeats until the process is killed when count exceeds 10. In order to get the asm-triggered interrupt to reach 10, I must use a loop.

4.4

xchildrennum() test: I copied most of my test from lab1 childrennum. I created a function that was simply an empty while(1), and used it to create 3 processes in main. Then I called childrennum() to check the resulting value, because I knew it was correct (and I also knew how many processes I had created anyways), and compared it to the return value of xchildrennum(). The output was as expected, so I could confirm the function was working.

xchprio() test: In order to test this, I first set the priority of one process, since chprio returns the *old* PID of a process. Then, I used xchprio() to update the priority of that process, and expected the previous priority to be returned, which it was. I repeated this a few times to confirm that xchprio() was working, and output was as expected, so I could confirm that.

xsleepms() test: In order to test this, I printed a message noting that I was starting to test xsleepms. This is important because I wanted to time the amount of time passing between the start and end of the test, and I needed to visually note when it started. Then, I passed 10000 as the argument to xsleepms(), as this would result in a 10 second sleep. After, I printed out a message that the test had ended. I expected a roughly 10-second wait between the start and end messages, which is what I observed, so I could confirm that everything was working.

Bonus: Completed