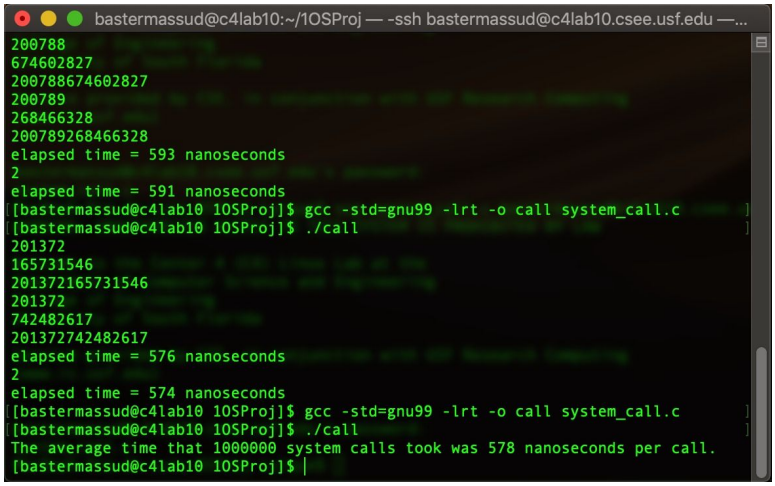
Luis Henrique Massud – U43063745
Anibal Jose Garcia McCausland – U74241605
February 1, 2019

Project 1: System Calls and Context Switch Measurements

Due date: Friday, February 1st, 11:59pm on Canvas

In this project, we measure the cost of both a system call and the cost of a context switch. To do this, we created programs which use different methods to measure the previously stated. To measure the system call, we simply repeated a system call, in this case read(0,0,0) and measured how long it took using clock_gettime(). To measure the cost of a context switch, we used the method that lmbench benchmark did. We ran two processes on one CPU and set up two pipes between them. Using these pipes, we write and read from process to process and then measure the cost using clock_gettime. Both programs were ran a large amount of time and the results averaged to get more accurate results. This problem is important because it is a good way to see the cost of these processes in a system.

Our approach was to make the programs as simple as possible. For the cost of a system call, we created a for loop to run a system call (read(0, 0, 0) large amount of times (1,000,000) and measured the time it took to run the loop using clock_gettime(CLOCK_MONOTONIC, &start) and clock_gettime(CLOCK_MONOTONIC, &end), start and end being a struct of type timespec which would give us the start and end time in seconds and microseconds using tv_sec and tv_nsec. We then calculated the cost of only the for loop running using the same method and then subtracted the two values (total time – loop time) to get a more accurate result of only the system call cost. We used nanosecond precision to get a more accurate and precise measurement. We initially had trouble using clock() and getting the time in microseconds because the precision was not high enough and ended up getting results like “0.0001 microseconds” for almost every trail. When switching to clock_gettime() and getting nanoseconds, it improved our results drastically.



Here we can see our debugging of our program. Initially we had multiple numbers printing to see what each one represented but in the last execution of the program we see only the final result which is accurate to +/- 4 nanoseconds.

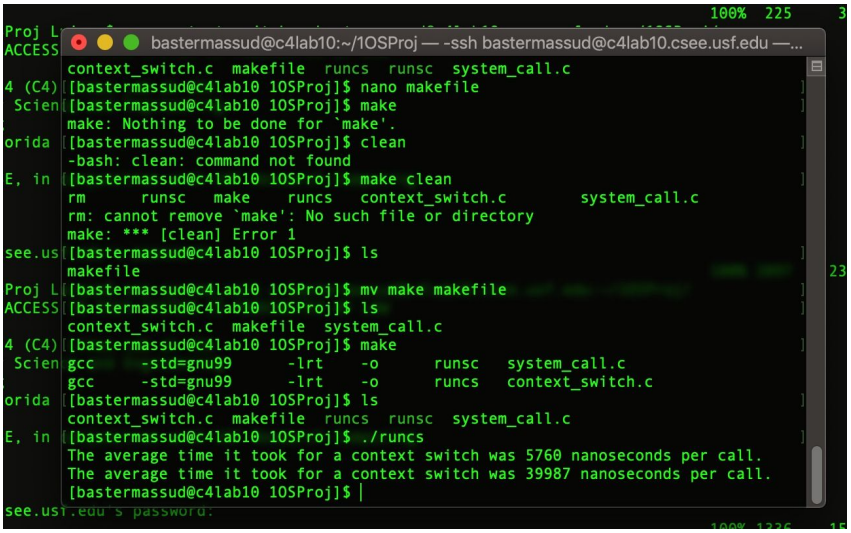
Luis Henrique Massud – U43063745
Anibal Jose Garcia McCausland – U74241605
February 1, 2019

Our approach for the cost of a context switch was ideally the same as for system call. We used the resources provided by the TA and professor to try and make our program as simple as possible. We struggled a lot with this measurement because though we were able to read and write from both processes, when reading from the parent process and printing what we read, we would not get a string as intended but an integer. We spent hours trying to find what was wrong because we had no complier error yet still not getting the desired results. While writing the report we had the idea to see what was being printed and we found a %d instead of a %s and this solved our issue. Another problem we encountered was trying to cycle back and forth multiple times between the parent and the child processes. Our thought process was to create a for loop to run x amount of times in for each process and so the steps would be as follows:

1. Process 1: writes
2. Process 1: blocked until process 2 writes
3. Process 2: reads
4. Print what was read
5. Process 2: writes
6. Process 2: blocked until process 1 writes
7. Process 1: reads
8. Print what was read
9. Repeat.

Our issue came when we tried to repeat the steps above. Instead of getting a print from each process interchanged, (P1, P2, P1, P2..etc) we would get x amount of prints from P2 and then x amounts of prints from P1. We are not positive as to if it is possible to do what we intended but we tried another way of implementing it as well. We tried to implement a pause() function to help alleviate our issue.
The pause function would be based on a signal that would tell the processor to resume playing only when the signal became 1. This idea did not really work the way we wanted and waited for the signal which was based on an user input. The main use of this is to create a way of addressing the CTRL+C command and initiating a functionality with it, not being suitable for our intended use.
We also had the issue of having to use sched_setaffinity() in order to run the two processes on a single GPU but this was easy to solve with a simple google search.
In addition, because of the fork() we had two results printing instead of one.

Luis Henrique Massud – U43063745
Anibal Jose Garcia McCausland – U74241605
February 1, 2019



Here we can see the results from our Context Switch Measurements. Here we see one of the issues we had which was getting two different measurements.

To conclude, even though we were not able to fix our issue from measuring the cost of a context switch, we measured the cost of one switch multiple times and gathered an average which helped the accuracy of our answer. All in all, this was an interesting project which helped us understand not only the concepts of what we learned in class but by implementing it we were able to clear up doubts we had originally had. We spent an estimated 15 hours working on this project.