John Gangemi – U68714612

Raj Patel – U69174395

**Project 1: System Calls and Context Switch Measurements**

**Overview:**

The purpose of this project is to measure the cost of a system call and the cost of a context switch.

Before we started to write any code we read over the project 1 pdf multiple times and highlighted what we believed to be key points. Also using that highlighted information we created some diagrams which are showcased at the end of the report in the diagrams section.

While coding we ran into many road blocks that we overcame using an iterative approach. Overall the linux programming manual was a very useful resource to find out more information regarding wait, pipes, read, write, gettimeofday, getuid() and fork.

It took us about 50 hours to complete the project. A lot of the time was spent in researching and reading the linux manual on the above listed topics.

**Methods we tried:**

1. **System Call-**

When we first started our original code we were getting a bunch of 0s and 1’s as the output for the individual cost of the system calls. For a while we were not able to figure out why. After a little bit of research we were able to conclude that Gettimeofday was serving as a function call as opposed to the a system call for the purposes of this lab and that is why we were constantly getting 1’s and 0’s. (could not provide example code because we scraped that code and started over)

After we found that out we looked into different system calls and decided to use **getuid** as our system call since it seemed to be the simplest one. We got it working with the user being able to enter the amount of system calls that he wanted to make. We wanted to increase the accuracy of the numbers and make sure we were obtaining consistent numbers so we added in a different number of trials on top of doing system calls to see if the numbers changed at all. After trying this method out we realized that this was redundant, since doing 1 trial and 500 system calls will give a number VERY close to doing 10 trails with 50 system calls a piece(as shown by the red highlighted sections above). After this we reverted back to the original method in which we just made a number of system calls and found the average of them all.

Below is a list of some of different trail and number of system calls we tried (showcasing the information is redundant)

|  |  |  |
| --- | --- | --- |
| Number of Trials | Number of System | Average time (microseconds) |
| 1 | 1 | 2.0 |
| 1 | 1,000,000 | .067942 |
| 1 | 50 | .1 |
| 1 | 500 | .072 |
| 1 | 10,000 | .0699 |
| 50 | 1 | .08 |
| 10 | 50 | .076 |
| 1 | 2 | 1 |

Below are the numbers we obtained once we measured just the number of system calls and obtained the average time they took. They seem to be converging around .065 microseconds.

|  |  |
| --- | --- |
| Number of System Calls | Average time (microseconds) |
| 1 | 2 |
| 2 | 1 |
| 3 | 0.667 |
| 4 | 0.5 |
| 5 | 0.4 |
| 10 | 0.2 |
| 25 | 0.12 |
| 50 | 0.1 |
| 75 | 0.093 |
| 100 | 0.09 |
| 200 | 0.075 |
| 400 | 0.0725 |
| 800 | 0.07125 |
| 1200 | 0.069167 |
| 2000 | 0.0725 |
| 4000 | 0.0695 |
| 10000 | 0.0686 |
| 50000 | 0.06824 |
| 100000 | 0.07081 |

1. **Context Switching-**

For a while we tried to measure context switches without using pipes. We thought we would be able to assume that the parent finished before the child process since it started beforehand but that is not necessarily a good assumption. After reading the linux manual on pipes we decided to take that route.

**Challenges we encountered:**

1. **System Call-**

How to correctly calculate the times in microseconds.

1. **Context Switching-**

We tried to **use sched\_setaffinity**. We actually spent close to 2 hours trying to do that and then we realized that fork actually does the same thing. According to the man page for **fork** when you fork something it makes sure that both the parent and the child processes are on the same CPU.

1. **Both Sections-**

**1)** One of the challenges we encountered is figuring out how to use gettimeofday. When we first read the linux manual on **gettimeofday** we thought that we were able to choose whether we wanted to use microseconds or seconds. So we just subtracted the microseconds we got very large numbers that seemed very unreasonable. Later re-reading the Linux manual page we realized that we can not just choose seconds or microseconds but rather we have to use both. We ended up just subtracting the difference between microseconds((converted to seconds) and the difference between seconds which were all added together and divided by the conversion factor . Below is the code showing how we did that.

// calculate system call execution time

double seconds = (double)end.tv\_sec - (double)start.tv\_sec;

double seconds\_from\_microseconds = ((double)end.tv\_usec - (double)start.tv\_usec) \* conversion\_factor;

double microseconds = (seconds + seconds\_from\_microseconds) / conversion\_factor;

**2)** Figuring out how to use the **pipes**

**3)** Figuring out where to place gettimeofday to get the correct time for context switches

**4)** After we completed the code one of the challenges we encountered was creating the tar files and make files.

**What we learned in the process:**

1. **System Call-**

* Properly using get timeofday to measure cost of system call

1. **Context Switching-**

* How to fork a process to create a child process
* How to create a pipe (open and close each end)
* How exit() works (had it in the code originally but ended up not using it)

1. **Both Sections-**

* How to properly use gettimeofday (where is should be placed)
* How to create and open a tar file
* How to create a make file
* Using the linux programming manual
* How to use getuid
* What belongs in a read me file
* How to use the terminal

**Which methods eventually worked:**

1. **System Call-**

The method that eventually worked was to use back to back system calls. Also we decided using just system calls instead of having number of trials and system calls since the numbers were so close making the information redundant. Also in the final method we used gettimeofday instead of clock\_gettime.

1. **Context Switching-**

Created two processes using fork. We didn’t need to use sched\_setaffinity because fork creates a child process with the same affinity mask as the parent process. Using a single pipe to connect the two processes.

**Diagrams:**

