Scott De Silva & Shaughn Seepaul
Operating Systems Project 1
DUE: 01/30/18 at 11:59 pm

Introduction:

In this project we were required to create two programs, system_call.c and context_switch.c, along with a makefile which was used to compile and run each file. In system_call.c, the time elapsed after a system call on a function had to be calculated using a method totally dependant on the user. Likewise, in context_switch.c, it measures the time it takes to do a context switch also dependant on the user. Using the hints and tips given to us by Dr. Iamnitchi, along with doing research online, we tried solving these programs.\

Approach:

SYSTEM_CALL.C:

In this program, we followed Dr Iamnitchi's tips by using a zero character read in a text file. The text file was created with read only permissions and a new one was created if none was present to be read. We tried a few functions in order to find the time elapsed. Firstly we tried using gettimeofday() which was a part of the sys/time.h library. However after calculating the time, we realised that it was not as accurate as it was supposed to be based on prior research. Therefore, we ended up using the clock_gettime function which was part of the time.h library which was found to be a bit more accurate in its readings. Two variables, 'start' and 'end', were created and the clock_gettime function was called on these two variables as shown:

```
clock_gettime(CLOCK_MONOTONIC, &start);     // current time at start
read(file_dscr,data,0);                     // system call to read file
clock_gettime(CLOCK_MONOTONIC, &end);       // current time at end
```

The difference was calculated between these two times and the average was found by repeating this process in a loop for 10000 times.

```
for(i = 0; i < SAMPLE_SIZE; i++)
{
  clock_gettime(CLOCK_MONOTONIC, &start);     // current time at start
  read(file_dscr,data,0);                     // system call to read file
  clock_gettime(CLOCK_MONOTONIC, &end);       // current time at end

  // finding the actual time to do the system call using the start and end time
  delta = end.tv_nsec - start.tv_nsec;
  sum += delta;
}
```

An example of running the code is demonstrated below:

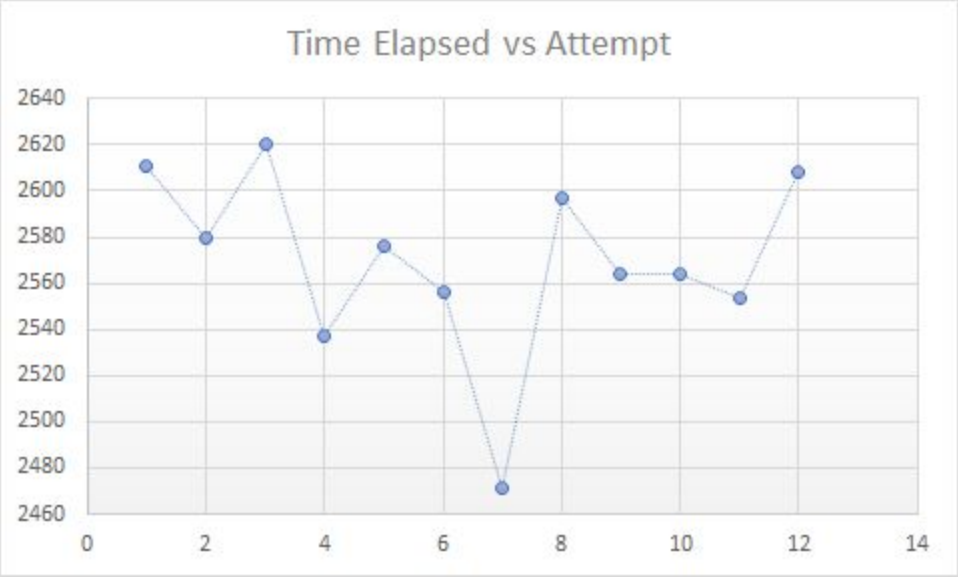
When the makefile is made, it compiles the code and also runs the output file in the same command. As we can see, the times varied for each execution (in this case the code was run three times). This was also seen when we ran the program on a different computer (a MAC). we obtained a lesser time and this could have been due to the MAC having a faster processor or was able to allocate more resources into calling the system call.

We also want to make note that the first time the program runs, the first calculation will always be greater than the others, by a noticeable amount, and this is due to the fact that every subsequent call accesses the desired data faster since it is now in cache.

```
[seepauls@c4cuda10 project1]$ make
gcc -o sc system_call.c -lrt
./sc
2519.124000 nanoseconds
gcc -o cs context_switch.c -lrt
./cs
AVG: 12756 nanoseconds
rm -f test.txt sc cs
[seepauls@c4cuda10 project1]$ make
gcc -o sc system_call.c -lrt
./sc
2545.691800 nanoseconds
gcc -o cs context_switch.c -lrt
./cs
AVG: 12495 nanoseconds
rm -f test.txt sc cs
[seepauls@c4cuda10 project1]$ make
gcc -o sc system_call.c -lrt
./sc
2537.497300 nanoseconds
gcc -o cs context_switch.c -lrt
./cs
AVG: 14137 nanoseconds
rm -f test.txt sc cs
[seepauls@c4cuda10 project1]$ make
gcc -o sc system_call.c -lrt
./sc
2461.255100 nanoseconds
gcc -o cs context_switch.c -lrt
./cs
AVG: 13691 nanoseconds
rm -f test.txt sc cs
```

12 values of the times obtained were recorded in a table and plotted on a graph:

Attempt	Time Elapsed (ns)
1	2610.913
2	2579.6787
3	2620.7753
4	2537.3024
5	2576.3912
6	2556.622
7	2471.4435
8	2597.3211
9	2563.626
10	2564.223
11	2553.5405
12	2607.8796



CONTEXT_SWITCH.C:

In this program, we decided to use the clock_gettime() function again because after reading a blog that juxtaposed the function to gettimeofday(), we decided clock_gettime() would provide more accurate results. After reviewing the background information Dr Iamnitchi included in the project file, we decided to do some more research to better understand the way pipes work in C and how we would be able to calculate the time difference during a context switch. The method of our approach was to get a timestamp right before the write to the first pipe in the parent process and the other right before the read of the first pipe in the child process. However, before even getting timestamps, we needed to bind the processor to a specific processor by using sched_setaffinity(). Then, we created two pipes which we saw as a parent pipe and a child pipe, corresponding to the parent process and the child process, respectively. After creating pipes, we called the fork() method to create the child process and we set up conditions to direct each process (and cater for chance that the fork failed). We coded routes for the process based on their process IDs. The code is full of comments describing the step by step process but the gist is as follows:

> The parent is responsible for sending out the first message to the child process through the first pipe, which we labeled “pp” for parent pipe. Then, we took a timestamp as soon as the message was sent out. Then, we set up a read from the child process to the parent, thus causing the parent process to be blocked and go into the child process. (Ideally, one would use wait(NULL); however, this caused our program to get stuck. The context switch was still able to execute based on our results so we ignored wait(NULL)) Meanwhile in the child process, the message from the parent is received through the “parent” pipe and a timestamp is taken as soon as the data is read because this marks the nearest point to the end of the context switch. We sent the child’s timestamp back to the parent process through the “child” pipe and once the parent process resumes, it will calculate the difference and add it to sum. We also included a printf() line to test our outputs and verify the data.

```
for(i = 0; i < SAMPLE_SIZE; i++)
{
  close(pp[0]);                                  //close read end of parent pipe before writing from it
  write(pp[1], " ", 1);                          //send arbitrary data to child via parent pipe
  clock_gettime(CLOCK_MONOTONIC, &start);        //get time closest to start of the context switch

  close(cp[1]);                                  //close write end of child pipe
  read(cp[0], &time_stmp, sizeof(time_stmp));    //read timestamp from child pipe

  delta = time_stmp - start.tv_nsec;             //(time child read pipe message) - (time parent sent message)
//   printf("pdelta:%ld\n", delta);               //use to check difference values
  sum += delta;
}
```

Figure above shows parent process loop of sending and receiving data via pp or “parent” pipe

Something we had a decent amount of trouble with was the formatting of the long value returned by the timespec struct used to implement the clock_gettime() function.

- Had to close ends of pipes
- Had to find bounds to avoid buffer overflow and getting negative numbers
- First number was always bigger because data is not in cache yet
    - We got at least one relatively large number compared to the rest of the dataset

```
[seepauls@c4cuda10 project1]$ make
gcc -o sc system_call.c -lrt
./sc
2519.124000 nanoseconds
gcc -o cs context_switch.c -lrt
./cs
AVG: 12756 nanoseconds
rm -f test.txt sc cs
[seepauls@c4cuda10 project1]$ make
gcc -o sc system_call.c -lrt
./sc
2545.691800 nanoseconds
gcc -o cs context_switch.c -lrt
./cs
AVG: 12495 nanoseconds
rm -f test.txt sc cs
[seepauls@c4cuda10 project1]$ make
gcc -o sc system_call.c -lrt
./sc
2537.497300 nanoseconds
gcc -o cs context_switch.c -lrt
./cs
AVG: 14137 nanoseconds
rm -f test.txt sc cs
[seepauls@c4cuda10 project1]$ make
gcc -o sc system_call.c -lrt
./sc
2461.255100 nanoseconds
gcc -o cs context_switch.c -lrt
./cs
AVG: 13691 nanoseconds
rm -f test.txt sc cs
[seepauls@c4cuda10 project1]$ make
gcc -o sc system_call.c -lrt
./sc
-97389.330000 nanoseconds
gcc -o cs context_switch.c -lrt
./cs
AVG: 12530 nanoseconds
rm -f test.txt sc cs
```

```
[seepauls@c4cuda10 project1]$ make
gcc -o sc system_call.c -lrt
./sc
2607.728200 nanoseconds
gcc -o cs context_switch.c -lrt
./cs
AVG: 12596 nanoseconds
rm -f test.txt sc cs
[seepauls@c4cuda10 project1]$ make
gcc -o sc system_call.c -lrt
./sc
2557.490500 nanoseconds
gcc -o cs context_switch.c -lrt
./cs
AVG: 12418 nanoseconds
rm -f test.txt sc cs
[seepauls@c4cuda10 project1]$ make
gcc -o sc system_call.c -lrt
./sc
2454.066300 nanoseconds
gcc -o cs context_switch.c -lrt
./cs
AVG: 13665 nanoseconds
rm -f test.txt sc cs
[seepauls@c4cuda10 project1]$ make
gcc -o sc system_call.c -lrt
./sc
2555.382200 nanoseconds
gcc -o cs context_switch.c -lrt
./cs
AVG: 13998 nanoseconds
rm -f test.txt sc cs
[seepauls@c4cuda10 project1]$ make
gcc -o sc system_call.c -lrt
./sc
2509.285100 nanoseconds
gcc -o cs context_switch.c -lrt
./cs
AVG: 14093 nanoseconds
rm -f test.txt sc cs
```

Several values of the times obtained were recorded and a graph was plotted against the amount of times recorded

Attempt	Time Elapsed(ns)
1	12529
2	12756
3	12495
4	14137
5	13691
6	12530
7	12596
8	12418
9	13665
10	13998
11	14093
12	14095
13	12194
14	12607
15	12555
16	12741