OPERATING SYSTEMS PROJECT 1

SYSTEM CALLS, PROCEDURE CALLS AND CONTEXT SWITCH MEASUREMENTS

BY

SAI KIRAN REDDY MALIKIREDDY(U63053381)

KIRANMAYI KARRI(U81100483)

Table of Contents

1. INTRODUCTION

This report gives a brief description of how we measured the cost of system calls, procedure calls and context switches. These measurements are implemented using c programming and tested on various system architectures such as Intel core I7(C4 lab machine), AMD opteron etc,. The detailed description of each execution and results are reported in further sections.

2. SYSTEM CONFIGURATION

Below are the system configurations of the machines on which we tested.

2.1 HARWARE ENVIRONMENT

Configurations	**AMD Opteron**	**C4 Lab**	**Intel Xeon(R)**	**Intel Core I7**
Processor	AMD Opteron	Intel(R) Core(TM) i7	Intel Xeon(R)	Intel(R) Core(TM) i7
Model	AMD Opteron QEMU	Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz	Intel Xeon(R)-E7310 @ 1,60GHz	Intel(R) Core(TM) i7-4770K CPU @ 3.50GHz
Cycle Time	2399.998MHz	3392.20MHz	1601.61MHz	3520.33MHz
RAM Size	8GB	3GB	115 GB	16GB
Number of Processors	4	8	8	8

2.2 SOFTWARE ENVIRONMENT

Operating System: Linux

Compiler : gcc

Language : C

3. COST MEASUREMENTS

3.1 SYSTEM CALL COST CALCULATIONS

Description

System call : It is a service request to an operating system by a user program and also an interface between the process and the operating system kernel.

How it works: Whenever a system call is requested by the user process, the processor will be switched to the privileged mode via a system call handler. Now the program counter is set to a address in the kernel space and the stack of the kernel is pointed by the stack pointer. The kernel identifies the type of service with the system call number(each system call has a specific number) and executes the service and returns. Then the processor will be switched to unprivileged mode.

Importance: In the programmer point of view, system calls are very essential functions. Programmers may not be concerned about the system calls as we have library functions which in turn invokes system call and performs the required functions. For example, if we a program wants to print something they can simple use printf() function which in turn invokes write() system call. Similarly, to change permissions of a file we have chmod functions.

Program Implementation

To measure the cost of the system call, we have experimented the program for two types of system calls "getpid()" and "setuid()" in a loop of 10000 iterations. To calculate the cost we executed the rdtsc() (Read time stamp counter) instruction in between the execution of the getpid() system call. For each iteration the clock cycles are calculated by accumulating the previous iteration cost with elapsed cost of the current iteration , here we excluded the overhead time (Refer Overhead Calculation). And finally the total cost is divided by the total number of iterations to derive the average cost for the system call.

Observations

During the experimentation for these measurements we have learned few interesting aspects.

RDTSC() : It is a time stamp counter which returns the number of CPU Cycles. In our program we observed a that the first execution of this call is returning more cycles when compared with the next executions. From our analysis we could see that this is because, for

the first call it will have an overhead to load the code into memory and then to cache and from the next execution it takes the data from the cache stored in the previous run.

Calculation of the overhead time : We performed this process using a different code which runs a loop of 10000 iterations which executes two rdtsc() functions with no other operations in the loop, from these results we calculated the total elapsed time. Then reported the overhead time using the elapsed time over number of iterations.

Exclusion of Overhead time : In the initial coding stage this hasn't been excluded from the total cost of the system call cost, later from our analysis and discussions we found that there is a need to exclude the execution time for the two rdtsc() calls in the code, so we have measured this overhead time and then excluded the resultant time from the actual result.

No loop Overhead : We could see there will be no loop overhead in the code as the measurements are calculated inside the loop.

Results

To obtain accurate results we have consider the statistics of atleast three runs and calculated the average of those.
The results below are represented in clock cycles. We have then converted the clock cycles to nanoseconds manually , they can found in the below table.

Overhead for rdtsc() in C4 lab machine

```
[malikireddy@c4lab11 project1]$ gcc -o om overheadmeasure.c
[malikireddy@c4lab11 project1]$ ./om
Average overhead in cycles:  28.028164
[malikireddy@c4lab11 project1]$ ./om
Average overhead in cycles:  28.973317
[malikireddy@c4lab11 project1]$ ./om
Average overhead in cycles:  28.388940
```

Overhead for rdtsc() on Intel core-i7

```
[sreddy_admin@infa-server proj1]$ gcc -o om overheadmeasure.c
[sreddy_admin@infa-server proj1]$ ./om
Average overhead in cycles:  26.034535
[sreddy_admin@infa-server proj1]$ ./om
Average overhead in cycles:  26.157300
[sreddy_admin@infa-server proj1]$ ./om
Average overhead in cycles:  26.148621
```

Overhead for rdtsc() on AMD opteron

```
[kiran@datanode_n0 project1]$ gcc -o om overheadmeasure.c
[kiran@datanode_n0 project1]$ ./om
Average overhead in cycles:  40.301487
[kiran@datanode_n0 project1]$ ./om
Average overhead in cycles:  40.018954
[kiran@datanode_n0 project1]$ ./om
Average overhead in cycles:  39.604379
```

System call cost on C4 lab machine

```
[malikireddy@c4lab11 project1]$ gcc -o sm systemcallmeasure.c
[malikireddy@c4lab11 project1]$ ./sm
Average cost in CPU cycles for getpid : 8.641674
Average cost in CPU cycles for setuid : 1057.376145
[malikireddy@c4lab11 project1]$ ./sm
Average cost in CPU cycles for getpid : 7.597450
Average cost in CPU cycles for setuid : 1011.564098
[malikireddy@c4lab11 project1]$ ./sm
Average cost in CPU cycles for getpid : 8.870237
Average cost in CPU cycles for setuid : 1036.267022
```

Overhead for rdtsc() on Intel xeon(R)

```
[kiran@hadoop-server project1]$ gcc -o om overheadmeasure.c
[kiran@hadoop-server project1]$ ./om
Average overhead in cycles:  70.864702
[kiran@hadoop-server project1]$ ./om
Average overhead in cycles:  71.627301
[kiran@hadoop-server project1]$ ./om
Average overhead in cycles:  71.911488
```

System call cost on AMD Opteron

```
[kiran@datanode_n0 project1]$ gcc -o sm systemcallmeasure.c
[kiran@datanode_n0 project1]$ ./sm
Average cost in CPU cycles for getpid : 43.090113
Average cost in CPU cycles for setuid : 1219.448012
[kiran@datanode_n0 project1]$ ./sm
Average cost in CPU cycles for getpid : 42.788501
Average cost in CPU cycles for setuid : 1218.983895
[kiran@datanode_n0 project1]$ ./sm
Average cost in CPU cycles for getpid : 42.219751
Average cost in CPU cycles for setuid : 1213.163701
```

System call cost on Intel core-i7

```
[sreddy_admin@infa-server proj1]$ gcc -o sm systemcallmeasure.c
[sreddy_admin@infa-server proj1]$ ./sm
Average cost in CPU cycles for getpid : 7.436612
Average cost in CPU cycles for setuid : 572.813347
[sreddy_admin@infa-server proj1]$ ./sm
Average cost in CPU cycles for getpid : 7.004577
Average cost in CPU cycles for setuid : 573.947889
[sreddy_admin@infa-server proj1]$ ./sm
Average cost in CPU cycles for getpid : 7.924875
Average cost in CPU cycles for setuid : 573.991457
```

System call cost on Intel Xeon(R)

```
[kiran@hadoop-server project1]$ gcc -o sm systemcallmeasure.c
[kiran@hadoop-server project1]$ ./sm
Average cost in CPU cycles for getpid : 16.372249
Average cost in CPU cycles for setuid : 1210.682247
[kiran@hadoop-server project1]$ ./sm
Average cost in CPU cycles for getpid : 16.837758
Average cost in CPU cycles for setuid : 1213.188735
[kiran@hadoop-server project1]$ ./sm
Average cost in CPU cycles for getpid : 16.605531
Average cost in CPU cycles for setuid : 1211.002457
```

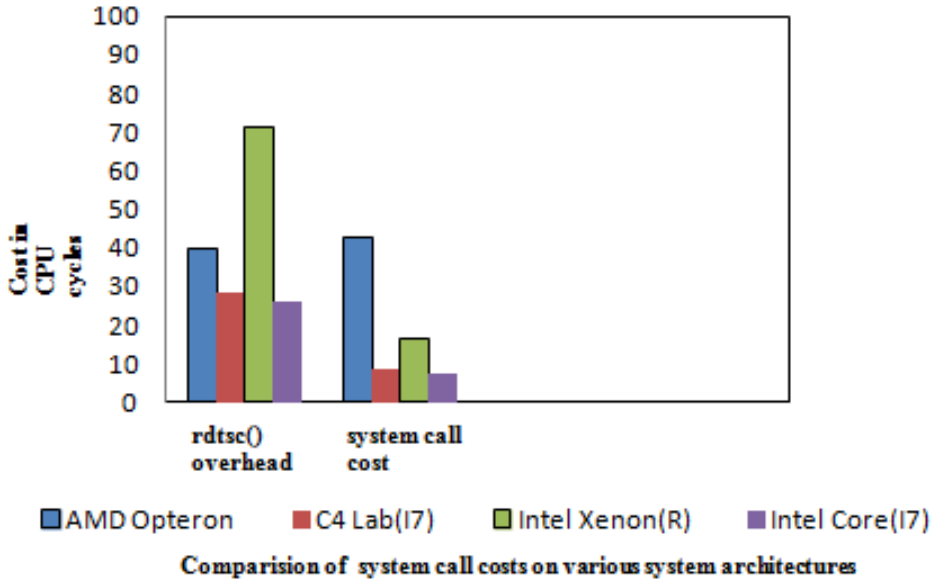
Table representing results in clock cycles and nanoseconds.

Conversion formula:

Time in nanoseconds = (1/processor speed(hertz))*number cycles

System Call	**Average Cost in CPU cycles**			
	AMD Opteron	**C4 Lab**	**Intel Xeon(R)**	**Intel Core I7**
	2399.998MHz (speed)	3392.20MHz (speed)	1601.61MHz (speed)	3520.33MHz (speed)
Overhead time for rdtsc()	40 cycles (16.64 ns)	28.50 cycles(8.38ns)	71.45 cycles(44.58ns)	26 cycles(7.384ns)
System call cost for getpid()	42.85 cycles (17.83ns)	8.37 cycles(2.42 ns)	16.41 cycles(10.23ns)	7.45 cycles(2.11ns)
System call cost for setuid()	1261.60 cycles(506.10ns)	1031.84 cycles(303.36ns)	1210.77 cycles(755.52ns)	573.86 cycles(162.98ns)

Graphs:



Comparision of system call costs on various system architectures

The above graph represents the results tested on various systems, the costs of the system call has variation on each different architecture due to the processor speed etc.,

Challenges

Why cost for system call is high for the initial run?

When we executed this code , we observed that the first execution of the system call is taking more time when compared to next executions. From our analysis , we found that for the first run the there might be some additional overhead as for the subsequent runs the system call is cached.

3.2 PROCEDURE CALLS COST CALCULATIONS

Description

Procedure call: It is a call to one of the function in the code. In other words, we can say that it is a call to the subroutine of the code which implements a specific function that may be executed several times from the various other functions in that code.

How it works: Whenever a procedure call occurs, the processor will trigger the function called and execute each and every instruction in that function and then return to the code where it was called and then process the next instruction.

Importance: The main advantage of this type of call is reusability.

Program Implementation

This is a simple code with a function call executed from another function. To measure the cost of the call , we implemented a function with a loop of 10000 iterations and have used rdtsc() before and after the call of the function. For every iteration, we calculated the elapsed time for every function call and accumulated the results for all the iterations, here we have excluded the overhead time of rdtsc() calculated previously. Then we got the average cost using this total cost over number of iterations.

For accuracy of the results , we have implemented in process considering various number of arguments ranging from zero to six.

Observations

System Call vs. Procedure Call : We observed from the results of the both call, that the system call is taking more cycles to execute when compared to the procedure call. The reason for this might be that system call has to switch between the user and system modes for their execution while the procedure call execution takes place in the user mode itself.

Results

To obtain accurate results we have consider the statistics of atleast three runs and calculated the average of those.
The results below are represented in clock cycles. We have then converted the clock cycles to nanoseconds manually , they can found in the below table.

Procedure call cost on AMD Opteron

```
[kiran@datanode_n0 project1]$ gcc -o pm proccallmeasure.c
[kiran@datanode_n0 project1]$ ./pm
Average cost for proc call with 0 args in cycles : 29.013755
Average cost for proc call with 1 args in cycles : 31.172416
Average cost for proc call with 2 args in cycles : 33.392014
Average cost for proc call with 3 args in cycles : 35.367510
Average cost for proc call with 4 args in cycles : 34.759265
Average cost for proc call with 5 args in cycles : 35.980674
[kiran@datanode_n0 project1]$ ./pm
Average cost for proc call with 0 args in cycles : 29.000758
Average cost for proc call with 1 args in cycles : 31.150885
Average cost for proc call with 2 args in cycles : 33.389372
Average cost for proc call with 3 args in cycles : 35.379426
Average cost for proc call with 4 args in cycles : 34.769124
Average cost for proc call with 5 args in cycles : 35.989112
```

Procedure call cost on C4 Lab machine(Inter core I7)

```
[malikireddy@c4lab11 project1]$ gcc -o pm proccallmeasure.c
[malikireddy@c4lab11 project1]$ ./pm
Average cost for proc call with 0 args in cycles : 5.582104
Average cost for proc call with 1 args in cycles : 7.541340
Average cost for proc call with 2 args in cycles : 7.100112
Average cost for proc call with 3 args in cycles : 7.481147
Average cost for proc call with 4 args in cycles : 7.059175
Average cost for proc call with 5 args in cycles : 6.483418
[malikireddy@c4lab11 project1]$ ./pm
Average cost for proc call with 0 args in cycles : 5.204357
Average cost for proc call with 1 args in cycles : 7.260781
Average cost for proc call with 2 args in cycles : 7.007513
Average cost for proc call with 3 args in cycles : 6.942045
Average cost for proc call with 4 args in cycles : 7.084633
Average cost for proc call with 5 args in cycles : 6.469172
```

Procedure call cost on Intel Xeon(R)

```
[kiran@hadoop-server project1]$ gcc -o pm proccallmeasure.c
[kiran@hadoop-server project1]$ ./pm
Average cost for proc call with 0 args in cycles : 9.380224
Average cost for proc call with 1 args in cycles : 11.238241
Average cost for proc call with 2 args in cycles : 11.229146
Average cost for proc call with 3 args in cycles : 11.873197
Average cost for proc call with 4 args in cycles : 13.378112
Average cost for proc call with 5 args in cycles : 13.443178
[kiran@hadoop-server project1]$ ./pm
Average cost for proc call with 0 args in cycles : 9.437514
Average cost for proc call with 1 args in cycles : 11.253697
Average cost for proc call with 2 args in cycles : 11.263110
Average cost for proc call with 3 args in cycles : 11.644027
Average cost for proc call with 4 args in cycles : 13.343224
Average cost for proc call with 5 args in cycles : 13.4370023
```

Procedure call cost on Intel Core I7

```
[sreddy_admin@infa-server proj1]$ gcc -o pm proccallmeasure.c
[sreddy_admin@infa-server proj1]$ ./pm
Average cost for proc call with 0 args in cycles : 3.220619
Average cost for proc call with 1 args in cycles : 3.859913
Average cost for proc call with 2 args in cycles : 4.349152
Average cost for proc call with 3 args in cycles : 7.032066
Average cost for proc call with 4 args in cycles : 5.538221
Average cost for proc call with 5 args in cycles : 5.509215
[sreddy_admin@infa-server proj1]$ ./pm
Average cost for proc call with 0 args in cycles : 3.251052
Average cost for proc call with 1 args in cycles : 3.851153
Average cost for proc call with 2 args in cycles : 4.341305
Average cost for proc call with 3 args in cycles : 7.075299
Average cost for proc call with 4 args in cycles : 5.602789
Average cost for proc call with 5 args in cycles : 5.638779
```

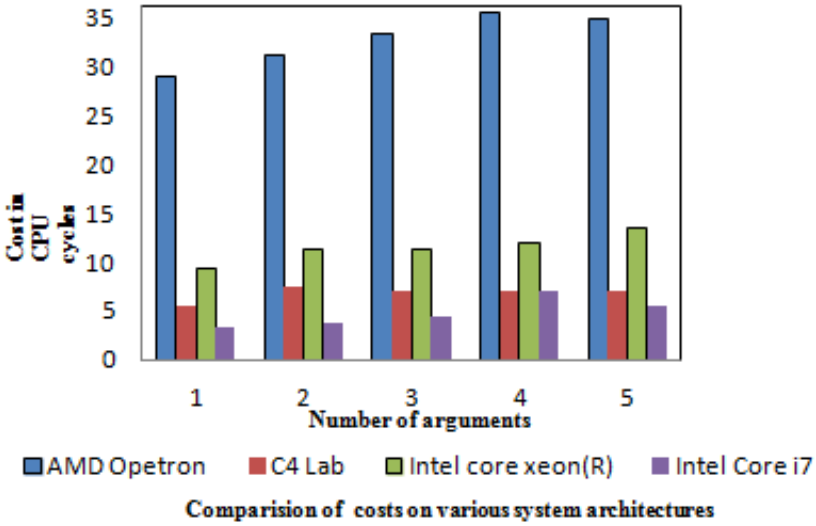
Table representing results in clock cycles and nanoseconds.

Conversion formula:

Time in nanoseconds = (1/processor speed(hertz))*number cycles

Procedure Call	**Average Cost in CPU cycles**			
	AMD Opteron	**C4 Lab**	**Intel Xeon(R)**	**Intel Core I7**
	2399.998MHz (speed)	3392.20MHz (speed)	1601.61MHz (speed)	3520.33MHz (speed)
Zero Arguments	29.00cycles(12.06ns)	5.46cycles(1.60ns)	9.40cycles(5.86ns)	3.24cycles(0.92ns)
One Argument	31.16cycles(12.96ns)	7.50cycles(2.20ns)	11.23cycles(7.00ns)	3.85cycles(1.09ns)
Two Arguments	33.38cycles(13.88ns)	7.01cycles(2.06ns)	11.24cycles(7.01ns)	4.34cycles(1.23ns)
Three Arguments	35.38cycles(14.71ns)	7.00cycles(2.05ns)	11.86cycles(7.40ns)	7.03cycles(1.99ns)
Four Arguments	34.76cycles(14.46ns)	7.02cycles(2.06ns)	13.36cycles(8.33ns)	5.51cycles(1.56ns)
Five Arguments	35.98cycles(14.96ns)	6.47cycles(1.90ns)	13.43cycles(8.38ns)	5.59cycles(1.58ns)

Graphs :



Comparision of costs on various system architectures

The above graph represents comparison between results of various system architectures for procedure calls.

Challenges

We observed a scenario when executing the procedure calls with various number of arguments that the number of cycles increasing along with the increase of the number of arguments. But the execution of procedure call with zero arguments is taking more time than few other procedure call with more number of arguments. This is because it is the first call in the code and it might have some additional overhead.

3.3 CONTEXT SWITCH COST CALCULATIONS

Description

Context Switching : It is a procedure of switching from one process to another.

How it works: Suppose there are two processes running, in case of context switching , the kernel suspends the first process stores the context (state of the process), then retrieves the details of the second process and restores that context in the CPU registers. It helps in enabling multiple process to share one CPU.

Why Context Switches are Important? Multitasking operating system has one of the important feature of context switching. Context switches allows these kind of operating systems create an illusion that it can execute multiple processes on a single CPU simultaneously. These context switches can also be implemented using various scheduling algorithms.

Program Implementation

We calculated the cost of context switch for both process and thread levels.

Context Switch using process: For this implementation ,we have created a new process using fork() and further used pipes to perform read and write operations between the child and parent process. We have forced the context switching to be performed twice in the process using a read operation performed by the parent and write by child process and then back to parent. The set affinity is being set in the code in order to execute the process on one processor. Also, we used waitpid() function in order to wait until the execution is complete. As we have performed two context switches in this implementation, in order to find the cost of one context switch , we took the average by dividing the result with 2. This process is repeated for 10000 iterations as done in the previous cost calculations. Note that we have excluded the overhead time for the creation of new process from the total cost.

Context Switch using threads: We have implemented this functionality using a clone () function, where we have using this to create a new thread and executed a function 'test' which performs a write operation. This process will execute the write function in its address space which was provided. As in this functionality we have context switch being invoked twice and hence divided the result with 2 to get the cost for only one context switch.

For accuracy, we have subtracted the overheads which we have obtained from the previous experiments.

Observations

Ensuring Accuracy: In order to maintain the accuracy of the cost , we have calculated the overhead time for creating a new process with fork and excluded this cost from the overall cost of each iteration.

Process vs. Threads : When the results of process and threads are compared we could observe that the cost for context switch in case of process in more than the cost for threads.

Results

To obtain accurate results we have consider the statistics of atleast three runs and calculated the average of those.
The results below are represented in clock cycles. We have then converted the clock cycles to nanoseconds manually , they can found in the below table.

Overhead time for creating a process(fork) and thread(clone) on AMD Opteron:

```
[kiran@datanode_n0 project1]$ gcc -w -o tm taskcreatemeasure.c
[kiran@datanode_n0 project1]$ taskset -c 0 ./tm
Time Taken for Fork is : 43280.849600
Time taken for clone is : 5286.211200
[kiran@datanode_n0 project1]$ taskset -c 0 ./tm
Time Taken for Fork is : 43880.963600
Time taken for clone is : 5352.266200
[kiran@datanode_n0 project1]$ taskset -c 0 ./tm
Time Taken for Fork is : 42806.366400
Time taken for clone is : 5306.244000
```

Overhead time for creating a process(fork) and thread(clone) on C4 Lab machine(I7):

```
[malikireddy@c4lab11 project1]$ gcc -w -o tm taskcreatemeasure.c
[malikireddy@c4lab11 project1]$ taskset -c 0 ./tm
Time taken for fork is : 28956.957600
Time taken for clone is : 4085.531200
[malikireddy@c4lab11 project1]$ taskset -c 0 ./tm
Time taken for fork is : 28819.843600
Time taken for clone is : 4091.799200
[malikireddy@c4lab11 project1]$ taskset -c 0 ./tm
Time taken for fork is : 28966.416400
Time taken for clone is : 4037.184400
```

Overhead time for creating a process(fork) and thread(clone) on Intel Xeon(R):

```
[kiran@hadoop-server project1]$ gcc -w -o tm taskcreatemeasure.c
[kiran@hadoop-server project1]$ taskset -c 0 ./tm
Time Taken for Fork is : 41546.992800
Time taken for clone is : 4546.459200
[kiran@hadoop-server project1]$ taskset -c 0 ./tm
Time Taken for Fork is : 40700.890800
Time taken for clone is : 4604.499000
[kiran@hadoop-server project1]$ taskset -c 0 ./tm
Time Taken for Fork is : 41442.591600
Time taken for clone is : 4566.209400
```

Overhead time for creating a process(fork) and thread(clone) on Intel Core I7:

```
[sreddy_admin@infa-server proj1]$ gcc -w -o tm taskcreatemeasure.c
[sreddy_admin@infa-server proj1]$ taskset -c 0 ./tm
Time Taken for Fork is : 27890.657600
Time taken for clone is : 3985.531200
[sreddy_admin@infa-server proj1]$ taskset -c 0 ./tm
Time Taken for Fork is : 28019.245800
Time taken for clone is : 3991.799200
[sreddy_admin@infa-server proj1]$ taskset -c 0 ./tm
Time Taken for Fork is : 27956.416400
Time taken for clone is : 3997.184400
```

Cost in CPU cycles for context switch on AMD Opteron:

```
[kiran@datanode_n0 project1]$ gcc -w -o csm contextswitchmeasure.c
[kiran@datanode_n0 project1]$ taskset -c 0 ./csm
Context Switch in Cycles for Process Level is = 52380.638200
Context Switch in Cycles for Thread Level is = 38458.346800
[kiran@datanode_n0 project1]$ taskset -c 0 ./csm
Context Switch in Cycles for Process Level is = 53280.538000
Context Switch in Cycles for Thread Level is = 39980.654400
[kiran@datanode_n0 project1]$ taskset -c 0 ./csm
Context Switch in Cycles for Process Level is = 52380.272400
Context Switch in Cycles for Thread Level is = 38250.266500
```

Cost in CPU cycles for context switch on C4 Lab machine(I7):

```
[malikireddy@c4lab11 project1]$ gcc -w -o csm contextswitchmeasure.c
[malikireddy@c4lab11 project1]$ taskset -c 0 ./csm
Context Switch in Cycles for Process Level is =  40391.638200
Context Switch in Cycles for Thread Level is =  37699.346800
[malikireddy@c4lab11 project1]$ taskset -c 0 ./csm
Context Switch in Cycles for Process Level is =  40272.538000
Context Switch in Cycles for Thread Level is =  37915.364400
[malikireddy@c4lab11 project1]$ taskset -c 0 ./csm
Context Switch in Cycles for Process Level is =  40195.272400
Context Switch in Cycles for Thread Level is =  37790.449800
```

Cost in CPU cycles for context switch on Intel Xeon(R):

```
[kiran@hadoop-server project1]$ gcc -w -o csm contextswitchmeasure.c
[kiran@hadoop-server project1]$ taskset -c 0 ./csm
Context Switch in Cycles for Process Level is = 46404.050600
Context Switch in Cycles for Thread Level is = 36987.546000
[kiran@hadoop-server project1]$ taskset -c 0 ./csm
Context Switch in Cycles for Process Level is = 46868.528600
Context Switch in Cycles for Thread Level is = 36504.741300
[kiran@hadoop-server project1]$ taskset -c 0 ./csm
Context Switch in Cycles for Process Level is = 46134.391100
Context Switch in Cycles for Thread Level is = 36639.589500
```

Cost in CPU cycles for context switch on Intel Core(I7):

```
[sreddy_admin@infa-server proj1]$ gcc -w -o csm contextswitchmeasure.c
[sreddy_admin@infa-server proj1]$ taskset -c 0 ./csm
Context Switch in Cycles for Process Level is = 39584.698200
Context Switch in Cycles for Thread Level is = 35489.896800
[sreddy_admin@infa-server proj1]$ taskset -c 0 ./csm
Context Switch in Cycles for Process Level is = 37858.538000
Context Switch in Cycles for Thread Level is = 34915.289400
[sreddy_admin@infa-server proj1]$ taskset -c 0 ./csm
Context Switch in Cycles for Process Level is = 38958.720400
Context Switch in Cycles for Thread Level is = 34790.999800
```

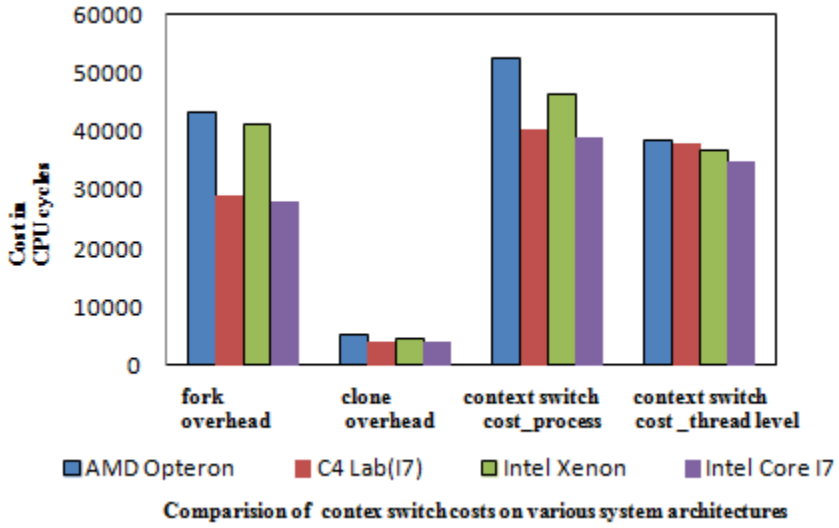
Table representing results in clock cycles and nanoseconds.

Conversion formula:

Time in nanoseconds = (1/processor speed(hertz))*number cycles

Context Switch	Average Cost in CPU cycles			
	AMD Opteron	**C4 Lab**	**Intel Xeon(R)**	**Intel Core I7**
	2399.998MHz (speed)	3392.20MHz (speed)	1601.61MHz (speed)	3520.33MHz (speed)
Fork_overhead	43280.2655cycles (18004.59ns)	28913.6666cycles (8500.618ns)	41230.1578cycles (25727.62ns)	27955.8686cycles (7939.467ns)
Clone_overhead	5270.6840cycles (2192.60ns)	4071.3284cycles (1196.97ns)	4572.1530cycles (2853.023ns)	3991.79920cycles (1133.671ns)
Context switch_Process Level	52380.7684cycles (21790.4ns)	40286.7684cycles (11844.31ns)	46468.6666cycles (28996.45ns)	38800.3642cycles (11019.3ns)
Context switch_Thread Level	38500.3366cycles (16016.14ns)	37801.3366cycles (11113.59ns)	36710.6389cycles (22907.44ns)	34900.5896cycles (9911.767ns)

Graphs:



The above graph represents results on 4 different architectures for overheads and context switch costs.

Challenges

It was a bit challenging for us to find the costs taken for context switch when compared to the previous executions in this project(syscall and procedure call). In order to set the processor to one core, we used setaffinity function. But we had a thought that including this part of code in our implementation makes only that part of the code to execute on the core, hence while executing the program we used a UNIX command "taskset" , so as to run the entire program one processor.

While executing the code, we need to prefix the 'taskset' command to the exec file, so that it sets the whole program to bound to one processor.

4. CONCLUSION

In this project, we have implemented functionalities to measure the cost of executing system call, procedure call and context switch on four different architectures. From our experiments we observed various interesting aspects such as cost of system call being more than the procedure call, results varying with vary in the architectures, increase in the number of arguments in procedure call results in increase of cost, cost of kernel threads context switch is more when compared to process.

5. REFERENCES

[1] https://www.ccsl.carleton.ca/~jamuir/rdtscpm1.pdf
[2] http://www.mcs.anl.gov/~kazutomo/rdtsc.h
[3] http://stackoverflow.com/questions/2668747/system-call-vs-function-call
[4] http://www.linuxquestions.org/questions/programming-9/what-are-the-drawbacks-of-syscalls-and-context-switches-257171/
[5] http://faculty.cs.niu.edu/~berezin/463/assns/clocks.html
[6] http://www.linfo.org/context_switch.html
[7] http://unix.stackexchange.com/questions/23106/limit-process-to-one-cpu-core
[8] http://www.personal.kent.edu/~rmuhamma/OpSystems/Myos/threads.htm
[9] http://www.gnu.org/software/libc/manual/html_node/CPU-Affinity.html