Measuring Context Switch Costs

Alexandre A. S Lopes ID: 301307001
Jay Maity ID: -301302461
Simon Fraser University
CMPT 886 - Assignment 1

I. Introduction

I/O requests, interrupts and other events can cause the operating system to change a CPU from its current task and to run a kernel routine. Such operation happens frequently on general-purpose systems. When an interrupt occurs, the system needs to save the current context of the process running on the CPU so that it can restore that context when its processing is done, essentially suspending the process and then resuming it. The context is represented in the PCB of the process; it includes the value of the CPU registers, the process state, and memory-management information. Generically, we perform a state save of the current state of the CPU, be it in kernel or user mode, and then a state restore to resume operations.[1]

II. Objective

The goal of this report is to explain the methodology, techniques and result of context switch time. First two experiments deal with function call time and last two experiments focussed on context switch time for processes and threads.

III. Methodology

We have used high resolution monotonic timer to measure all the experiments, since our goal is to measure the elapsed time this timer is the more suitable. Clock itself taking some time to execute. To be more accurate, we have measured two clock functions execution time by putting no code between them and subtract those time from experimental results.
The idea behind using semaphore and pipes is, we can get the measurement using two different method to find out correctness of our methodology.
Also was used the same methodology to measure process and thread context switch since the two procedures can be measured using the same approach, which theoretically will lead to high accuracy when comparing the final results.

IV. Pre Measurement Procedures

Prior to execute the measurements two procedures were adopted in order to improve the fidelity of the measurement results.
First was generated a function called "**SetProcessMaxPriority()"** that is responsible for set up the Process and Thread priority of process for the highest priority level allowed by the OS (operation system) trying to avoid any interruption in the measurement process which can result in an inaccurate and imprecise result.
Second was generated a function called "**WarmCPU()"** the idea behind this function is to avoid the Dynamic frequency scaling which is a popular technique in computer architecture employed in the modern processors design "Whereby the frequency of a microprocessor can be automatically adjusted "on the fly", either to conserve power or to reduce the amount of heat generated by the chip."[4] this function is generating CPU overload through the execution of successive mathematical operations in order to increase the CPU clock thus allowing the measure procedures to reach the CPU in its maximum clock frequency.
Moreover in order to measure the process and thread context switch was used a library called Portable Hardware Locality (hwloc) this library is in charge to force our processes and threads to be running in the same cpu which is necessary for the experiments.

V. Measurement Procedures

For each experiment, the operation was repeated one million times and then the average was taken as well as the smaller time spent of this particular experiment measured.

1. To find the time for an empty function we put a timer before and after the function call and measured the time we are then .
2. The above strategy is used to measure getPid() function as well.
3. In addition was measured the time of the NOP (No operation) opcode which is the lowest expansive operation in the level of CPU for comparisons, again the above strategy was used.
4. To measure context switch time for processes, we have used pipes. We write one end of a pipe and wait for another process to write back to another pipe. The time between these procedures (excluding reading and writing) is measured as process switch time. We also have used two semaphore to find context switch time. This is done by setting semaphore value by one process and decrease the value by other.

5. To measure context switch time for threads, the same methodology was used.

VI. Function Calls Measurement

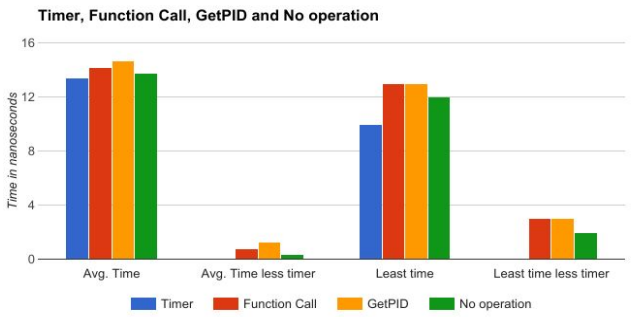
We have experimented with three procedures

1. Void function call
2. getPid system function call
3. NOP operation opcode

In all cases the cases, we have noted down Average time, Average time by subtracting the timer time, smallest time taken for any instance with timer time and smallest time taken without the timer time.

Description	Timer	Function Call	GetPID	No operation
Avg. Time	13.4	14.19	14.69	13.76
Avg. Excluding timer time	0	0.75	1.25	0.32
Least time	10	13	13	12
Least time excluding timer time	0	3	3	2

The results are shown below in graphical format.



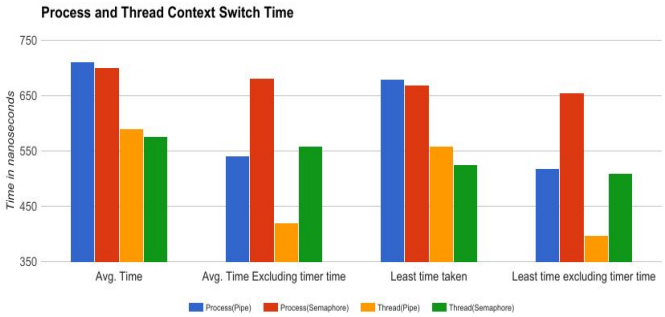
VI. Inter process and thread context switch

Inter process and thread context switch time is measured using two methods

1. Pipes
2. Semaphore

Above procedures resulted slightly different timing when we average them down. But for for least time measurement, it is taking a more time while using pipes.

Description	Process(Pipe)	Process(Semaphore)	Thread(Pipe)	Thread(Semaphore)
Avg. Time	710.9	701	590	577
Avg. Time Excluding timer time	541	682	421	558
Least time taken	680	670	559	525
Least time excluding timer time	518	655	397	510



VII. Conclusion

Experiments above brings some interesting results related to timer and use of different methods for context switch.

1. Time required for executing getpid and void function is not significantly different even if compared with NOP.
2. Unless we are subtracting the time required for timer, actual time for execution of the functions and NOP largely is incorrect.
3. Process context switch time is slightly greater (around 10-15% as per our experiment) than thread switching time.
4. Context switch time is a bit less when we are using semaphore. The possible reason is that despite all the effort to avoid interference in the measurement process the pipe is yet introducing a small overhead in the measurement.

References

[1] Silberschatz, Galvin, Gagne. *Operating System Concepts Essentials.*

[2] Andrew S. Tanenbaum. Herbert BOS. *Modern Operating Systems.*

[3] https://www.open-mpi.org/projects/hwloc

[4] https://en.wikipedia.org/wiki/Dynamic_frequency_scaling