Roy Chan
chanroyc@sfu.ca
301202770

CMPT 300 Assignment 2 (Part 1) Report

NOTE: All of the below are run on *Processor 1* of the multi-core CSIL machines by setting the affinity.

- 1: *part1.c* includes four of the following high resolutions timers:
  (Refer to *part1.c* of this assignment for comments.)

  Real-time Clock: Provides access to the system-wide realtime clock. Values are dynamic and when user changes system time, so will this timer. This timer is used to measure system-wide time, or in otherwords, "Time-of-day" time.
  Monotonic Clock: Absolute elapsed wall-clock time since a fixed point. This timer is not affected by "time-of-day" clock. It is used to measure elapsed time between two events.
  CPU time Clock: High resolution timer provided by the CPU for each process. Time spent waiting (sleeping process) is not counted. This timer measures CPU time consumed by a process.
  CPU Thread time clock: High resolution timer provided by the CPU for each thread. Time spent waiting on threads is not counted. This timer measures CPU time consumed by a thread.

- 2: *part2.c* measures the cost of a minimal function call using the *Monotonic Clock*.
  The function that is called in *part2.c* is *nothing()*, which returns nothing. In Figure 1., the average cost of of the minimal function call is 193 ns. This is to be expected as there is nothing being done between the start and stop time of the timer other than returning to its caller.

  Figure 1. Measurement of the cost of minimal function call.

Iteration	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Cost (ns)	101	238	220	231	206	132	205	181	191	225

  Average: 193 ns

- 3. *part3.c* measures the cost of a minimal system call using the *Monotonic Clock.*

  The function that is called in *part3.c* is *something()*, which calls *getpid()*. Prior to computing the system call, I hypothesized that a minimal system call would definitely cost more than the computation in part 2. Figure 2 contains the computations done by *part3.c* with the average cost being 2060.4 ns, 10 times larger than the above.

  Figure 2. Measurement of the cost of minimal system call.

Iteration	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Cost (ns)	2175	2431	2094	2080	2090	2160	2017	2050	1821	1686

  Average: 2060.4 ns

Roy Chan
chanroyc@sfu.ca
301202770

- 4. *part4.c* measures the cost of process switching. As well as the two above, I have chosen to use the monotonic timer to measure the elapsed time between each process switch. I initialized two different variables, *a* and *b* to be read and written into the pipes. I have placed the timers in in the parent process for each r/w. I noticed that the values for the first iteration of values were significaly larger (by a factor of <10), and dramatically decreased for the next 9 iterations of values (Figure 3). From this, I can conclude that after the first iteration, it utlized cachable memory regions to speed up the process switching, thus diminishing the amount of time it required to switch between the child and parent. Begins slow, speeds up and converges to 6000's ns and 700's ns r/w, respectively. The Average of the of r/w are 12504 ns and 941 ns, slightly higher than expected due to the startup. The startup could possibly be framed as an outlier as in more practical situations, processes are most likely already cached and ready for use.

Figure 3. Measurement of the cost of process switching.

Iteration	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Read (ns)	59512	10624	7665	6905	6976	6813	6863	6284	6294	6633
Write (ns)	2396	877	805	749	746	771	777	794	744	759

Average Read: 12504 ns
Average Write: 941 ns

- 5. *part5.c* Measures the cost of thread switching. As the rest of the assignment, I chose to use the monotonic timer to measure the elapsed time between each thread switch. As in Figure 4, the values or more or less settled in the high 400's ns. Instead of using 2 threads, I have also commented out 8 more threads in *part5.c* to test values. I have implemented the timer to begin after the code runs into the critical section, and stops as the next thread attemps to enter. As a result of my implementation and placement of the timer, the first value of the timer results in an insanely large value which is neglected/not used in my data set. As expected, thread switching is faster than process switching. The average time for thread switching is 462.2 ns.

Figure 4. Measurement of the cost of Thread switching.

Iteration	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Thread Switch Time (ns)	484	464	470	459	455	461	447	461	470	451

Average: 462.2 ns