Jerry Chien-Yu Chen
Oct. 30/ 2016

CMPT300
MEASURING CONTEXT SWITCH COSTS
Assignment 02

Test Points	Cost
For loop overhead with 100000000 iterations(CALLREPEAT)	1.762589 nano seconds
For loop overhead with 100000 iterations(FORKREPEAT)	1.690810 nano seconds
Read & Write overhead with FORKREPEAT iterations	462.343530 nano seconds
Average cost of min function call with CALLREPEAT iterations – for overheard	0.829867 nano seconds
Average cost of min system call with CALLREPEAT iterations – for overhead	0.908742 nano seconds
Average cost of MONOTONIC process switch with FORKREPEAT iterations	5544.401970 nano seconds
Average cost of PROCESS_CPUTIME process switch with FORKREPEAT	3745.017730 nano seconds
Average cost of 1 process switch with FORKREPEAT iterations	889.692120 nano seconds
Average cost of MONOTONIC thread switch with FORKREPEAT iterations	814.797260 nano seconds
Average cost of PROCESS_CPUTIME thread switch with FORKREPEAT	235.228210 nano seconds
Average cost of 1 thread switch with FORKREPEAT iterations	579.569050 nano seconds

COST OF MINIMUM FUNCTION CALL

Measuring the cost of a bare function has to be implemented carefully because it invokes the overhead of the resolutions clocks and the for loop. I chose to use the CLOCK_PROCESS_CPUTIME_ID clock because I specifically want the time of this specific function call running on the core without any other processes interrupting and be taken into account to the cost I am measuring. At first, I measured the cost with 100000000 iterations by putting the resolution clocks within the for loop. The output turned out to be a big number because the resolution clock overhead dominated the overall cost. Therefore, I decided to take the resolution clocks out of the for loop to minimize the dominance of the resolution clocks. The output did decrease a lot after doing so. And eventually I subtracted the for loop overhead from the output previously resulting the cost of a minimum function call to be 0.829867 ns.

COST OF MINIMUM SYSTEM CALL

Measuring the cost of a minimum system call has to be implemented carefully as well because it also invokes the overhead of the resolutions clocks and the for loop. After realizing the dominance of the overheard of resolution clocks previously, I measured the cost with 100000000 iterations by putting the resolution clocks outside the for loop just so the resolution clock overhead does not take over the cost. And I chose to use the CLOCK_PROCESS_CPUTIME_ID clock the same reason as I elaborated

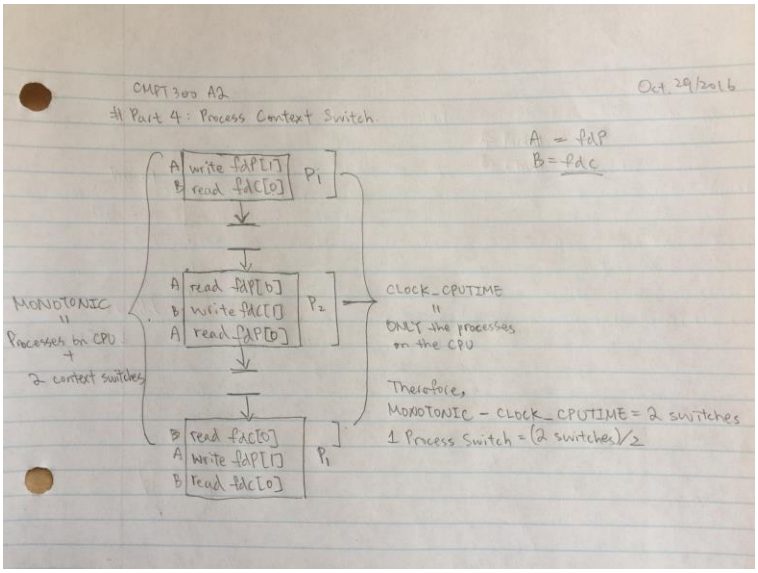
previously in the function call section. The output before subtracting out the for loop overhead was roughly about 2 ns. Eventually I subtracted the for loop overhead from the output previously resulting the cost of a minimum system call to be 0.908742 ns

Conclusion On Function Calls and System Calls

> After measuring the cost of both the minimum function call and minimum system call under 100000000 iterations, we can observe that the cost of the system call is greater than the cost of the function call on average. This is theoretically correct because calling a system call will need the system to enter the kernel mode to do so which would take longer time to process.

COST OF PROCESS SWITCHING

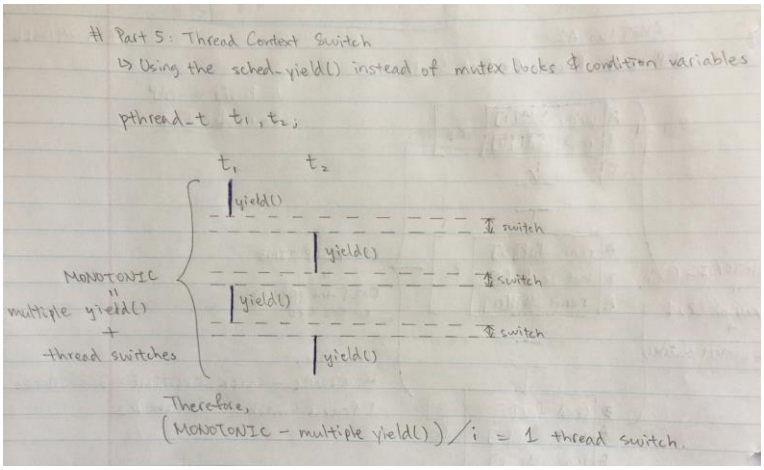
Measuring the cost of a process switch was not as easy as I thought. You have to be careful about where you are putting the clocks, and what is being measured with the type of clock you are using, as well as the overheads. Before I actually implemented the code, I drew this just to clarify the measure methodology.



Basically I first used the CLOCK_MONOTONIC clock to measure the time it takes for 2 pipes to pass the message back and forth for FORKREPEAT = 100000 times. I put the start clock just before write in the parent process and the stop clock after read in the parent process. As shown above on the left hand side of the figure, placing the clocks this way includes the measurement from write fdP[1] to read dfC[0] including 2 context switches. But this is not the measurement we want yet! We need to subtract out the time it takes for the process running on the core. Therefore, I do the same thing as above but this time with the CLOCK_PROCESS_CPUTIME_ID clock to obtain the time it takes for the process running on the core excluding 2 context switches. After all, I take the measurement that we got from using the CLOCK_MONOTONIC minus the measurement that we got from using the CLOCK_PROCESS_CPUTIME_ID clock giving us the time it takes to do 2 process context switches. And eventually I just divide the previous output by 2 giving us the cost/time it takes for one single process context switch! The result of a single process context switch turned out to be roughly 889.692120 ns. For detailed output results, please refer to the table above.

COST OF THREAD SWITCHING

Measuring the cost of a thread switch follows the same measure methodology I used previously in the process switch section which is using the CLOCK_MONOTONIC clock to take the overall time and then subtract out the time we don't need for FORKREPEAT = 100000 times. Instead of using the mutex locks and condition variables mentioned in the assignment pdf, I innovatively chose to use the sched_yield() function on 2 threads created instead. Basically what this function does is when the first thread on the run queue done processing, it gets put back to the end of the run queue. Since we only have 2 threads created, calling the sched_yield() function would result the threads switching back and forth inside the run queue with the thread context switches in between. The figure below illustrates the idea of my measure methodology:



Instead of having thread 1 change the integer sum value from 1 to 0 and having thread 2 change the integer sum value from 0 to 1, I only created one callback function for pthread_create() that has only sched_yield() inside the callback function. Because it is not so important what the threads themselves are actually doing inside the callback function since we are only measuring the thread context switch time. At first, I used the CLOCK_MONOTONIC clock to measure the time it takes from the beginning where pthread_create(t1) is to pthread_join(t2, NULL). This measurement obtains the thread running time and thread context switch time. Doing so, we obtain the overall time cost, and then we just need to subtract out the thread running time in order to get the time/cost of the multiple context switches.

Realizing that the thread running time here is actually just the overhead of sched_yield that was being ran by the callback function of the threads. So we use the measurement we obtained above subtract the overhead of sched_yield(), then we get the cost/time of a thread context switch! The result of a single process context switch turned out to be roughly 579.569050 ns. For detailed output results, please refer to the table above.

Conclusion On Process Switch and Thread Switch

> After measuring the cost of both the process context switch and thread context switch under 100000 iterations, we can observe that the cost of the process context switch is greater than the cost of the thread context switch on average. This is theoretically correct.

TESTING RESULTS ON CSIL WORKSTATION

```
File  Edit  View  Search  Terminal  Help
cyc42@asb9838nu-c06:~/sfuhome/cmpt300/a2$ make
gcc -pthread -std=gnu99 -pthread -std=gnu99   -c -o main.o main.c
gcc -pthread -std=gnu99   main.o   -o main
cyc42@asb9838nu-c06:~/sfuhome/cmpt300/a2$ make run
./main
CALL for loop overhead:1.762589 nano seconds
FORK for loop overhead:1.690810 nano seconds
Read and Write overhead:462.343530 nano seconds
Average cost of minimal function call:2.592456 nano seconds
Average cost of minimal funcCall minus forloop overhead:0.829867 nano seconds
Average cost of minimal system call:2.671331 nano seconds
Average cost of minimal sysCall minus forloop overhead:0.908742 nano seconds
Please send a message
child sends me: c
start calculate the clock difference of forking over here!!!
Average cost of monotonic process context switch:5544.401970 nano seconds
Please send a message
child sends me: c
start calculate the clock difference of forking over here!!!
Average cost of CPUTIME process context switch:3745.017730 nano seconds
Average cost of 2 process context switches:1799.384240 nano seconds
Average cost of 1 process context switch:899.692120 nano seconds
Average cost of monotonic thread context switch:814.797260 nano seconds
Average cost of CPUTIME thread context switch:235.228210 nano seconds
Average cost of 1 thread context switch:579.569050 nano seconds
cyc42@asb9838nu-c06:~/sfuhome/cmpt300/a2$
```

TESTING ENVIRONMENT