Performance Measurement for Processes and Threads

K. M. Sabidur Rahman

@01343723, UTSA

Email: wmr727@my.utsa.edu

Abstract

This project is to be submitted for Fall 2012 Operating Systems course instructed by Dr. Dakai Zhu.[1] The target of the project is to learn and practice system call, process, threads and synchronization.

1. Introduction

A process is a running program. OS creates process control block (PCB) for every process. Processes are independent units with their own address spaces, state information. Processes may interact with each other via inter-process communication mechanisms. On the other hand, a thread is an activity within a process. A single process may contain multiple threads. Threads are a vital way of using parallelism. While a process has its own address space, text, state information and other resources, threads within a process share address space, codes, global variables/data, open files and other resources. Threads have separate register status, stacks and private data. This mechanism allows threads to read from and write to the same data structures and variables, and also facilitates the smooth communication between threads. Inter-process communication (IPC) – is quite difficult and resource-intensive. Due to the structural and behavioral difference between processes and threads, the system requirements and performance of a particular workload done by a group of processes and the same workload done by a group of threads varies significantly. Through this project, we intend to see the implementation of these concepts.

2. Implementation Issues

2.1 System timers

In this project, measure the resolution of System time, gettimeofday() is used. It takes a structure type timeval variable as parameter and returns time of the day in that structure. A timeval has two components, both ints. One (called tv_sec) is exactly the value that would be returned by time, the time in seconds since 1/1/1970. The other (called tv_usec) is the number of microseconds into that second. [2]

In this project, to measure elapsed system time on a work, one 'timestamp' is taken before the start of the processing the workload and another 'timestamp after the end of the processing. Difference between the end and start timestamp is the time needed for the whole task.

2.2 Systems calls

In Part 2 of the project, fork() is used to create new child process. fork() creates an exact duplicate of the process that called it and returns the child id in the parent process and returns 0 in the child process.

In Part 3, pthread_create() is used to create a child thread. pthread_create() takes a function pointer and thread id as parameter. The function which is passed as parameter is the code segment for the newly created thread.

2.3 Inter-process communication

Shared memory is used to implement IPC between parent process and the child process. Shared Memory is an efficient means of passing data between programs. One program will create a memory portion which other processes (if permitted) can access. Here, the process creates a shared memory segment using shmget(). Once created, a shared segment can be attached to a process address space using shmat().[3] It can be detached using shmclt().[5] A shared memory segment is described by a control structure with a unique ID that points to an area of physical memory. The structure definition for the shared memory segment control structures and prototypes can be found in <sys/shm.h>.

2.4 **Synchronization mechanism**

Semaphore is a synchronization tool without busy waiting. It avoids busy waiting by blocking a process execution until some condition is satisfied. In this project, to maintain the synchronization among parent process/thread and child process/thread Semaphores are used. The function semget() initializes or gains access to a semaphore. [4] Conceptually a semaphore has an integer value and two indivisible (atomic) operations. The following sembuf structure specifies a semaphore operation, as defined in <sys/sem.h>.

```
struct sembuf {
      ushort_t    sem_num;   /* semaphore number */
      short       sem_op;    /* semaphore operation */
      short       sem_flg;   /* operation flags */
};
```
[6]

3. Process vs. Thread Performance Issue

For the performance measurement of processes at first a data file containing one million double values is created.

Then in the program, a child process is created. The parent process repeatedly does the following: read in X data values to the child process and wait for the result data from the child process. Once the parent process receives the result X data, they are written into another output data file.

In the child process, it first waits for the new batch of X data values. Then, for each of the X data value it performs some CPU intensive operations (in the experiment 1000 rounds of addition and subtraction operation is performed). Once the computation is done the child process signals the parent process.

The same way in Part 3, a child thread is created, instead of a child process. The code segment for the child is exactly same for both the case.

In this experiment two semaphores are used for the synchronization purpose, they are:

input: data value available for processing

output: result value is available for saving

Table 1 and 2 contains the pseudo code of the parent and child process/thread with the synchronization mechanism respectively.

```
While (file is not empty)
      Read X data values
      Signal(input)
      Wait(output)
      Write X result values
End
```

Table 1: Parent pseudo code

```
While (1)
      Wait(input)
      Perform 1000 round of operation on X data values
      Signal(output)
End
```

Table 2: Child Pseudo Code

4. **Discussion on run results**

As first part of the project, the value of the system timer is taken using gettimeofday() function multiple times consecutively without other statements in between. Then, the smallest value between two consecutive values is reported as resolution of the system. In this work resolution is found to be 1 microsecond.

For the second part of the project, processing time of the processes and threads on the generated workload for different batch size (X) is measured. Figure 1 shows the processing time requirement vs. shared memory size X (no of data value) for both the process and thread with operation size Y = 1000.

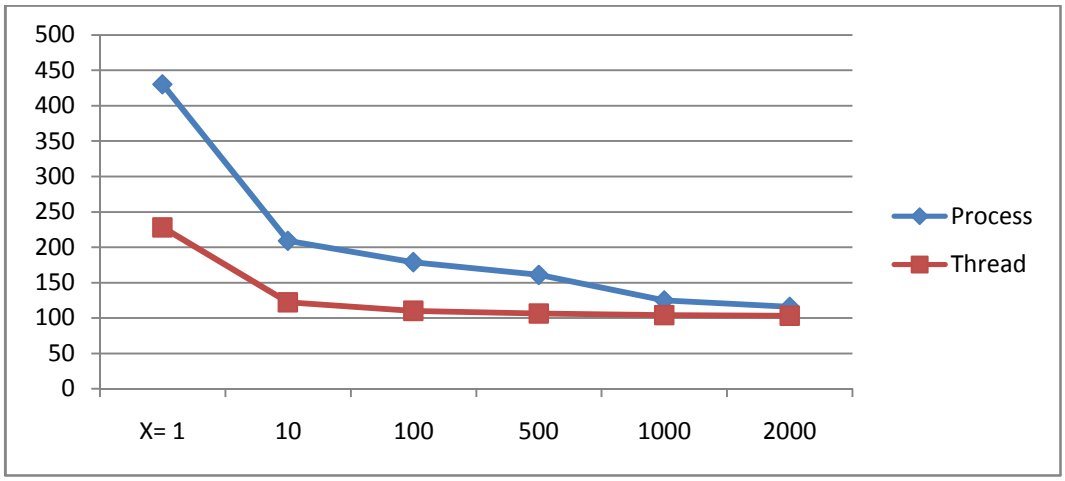

Figure 1: processing time vs. Shared Memory size

From the above figure we can see that the processing time for threads is lower than the processing time of processes while the batch size is small and the reason behind this phenomenon is that frequent communication between processes dominate the processing time as the communication overhead is high for two processes than two threads. But for large enough batch size processes outperform threads in terms of processing time. This is obvious because communication between parent thread and child thread dominates the processing time in case of bigger batch size. While using processes for large batch size, parent and child process works individually and require less communication and performs better than thread pairs.

So, it depends on the type of the application whether we should use processes or threads. From the experiment we can say that thread should be used for small task requiring frequent communication with parent process/thread and different process should be used for less communication requiring relatively big tasks. And this is the case for C thread and processes. In case of java it will be dependent on how JVM treats threads and processes inside it.

5. **Extra Implementation**

In case of multiple reader thread on the same Shared Memory, the performance does not improve as expected. Because, they all are competing for the same resource.[7] As Figure 2 suggests, the single thread is better if we do not increase the number of shared memory.

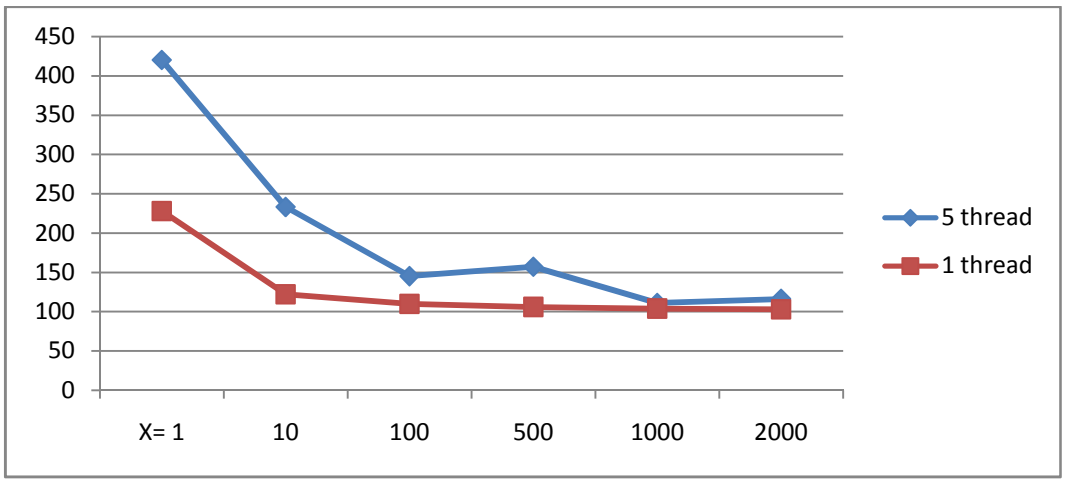

Figure 2: processing time vs. Shared Memory size

As the size of the shared memory increases, the performance gets better. This problem can be solved by implementing "Producer - Consumer Problem". Instead of a single shared memory, we need multiple shared memories for multiple threads to run work on. The synchronization between them will be solved with the solution concept of "Producer - Consumer Problem".

6. System Configuration

The following system is used for the performance measurement of processes and threads.

- Operating System : Ubuntu 12.04 LTE x64
- Hardware

  Memory: 6GB

  Processor: Intel(R) Core i3 Next Gen 2.2 GHz

7. **File Description**

createdatafile.c – Creates 1 million double values.

pro1part1.c- computes resolution

pro1part2.c – Process implementation

pro1part3.c – Thread implementation

pro1extra5.c – Implementation with 5 Child thread

commands.txt – Necessary commands to run C files on Linux.

8. Conclusion

This project has been a convenient away of learning implementation of system calls, process. thread and synchronization between them. The author would like to thank instructor Dr. Dakai Zhu for assigning such an effective project.

9. References

[1] http://www.cs.utsa.edu/dzhu/cs5523/12-proj-1.pdf

[2] http://rabbit.eng.miami.edu/info/functions/time.html

[3]http://www.lainoox.com/tag/shmat-example/

[4]http://cboard.cprogramming.com/c-programming/135151-putting-struct-into-shared-memory.html

[5]http://www.cs.cf.ac.uk/Dave/C/node27.html

[6] http://www.cs.cf.ac.uk/Dave/C/node26.html

[7]http://tldp.org/LDP/lpg/node53.html