

Threads and Processes Experiment

Manuel Gonzales

COMP 6D

British Columbia Institute of Technology

COMP 8005

Aman Abdulla

January 18, 2016

Contents

Summary	3
Introduction.....	4
Theory Behind the Experiment.....	4
Experimental Apparatus and Procedure.....	5
CPU Intensive.-	5
Disk I/O Intensive	5
Results and Discussion	6
Threads.....	9
Processes	10
Conclusion	12

Summary

A short experiment was conducted to compare the speed and efficiency of threads and processes in order to find which one is better. The tests were run on two different systems both using the Fedora 22 OS. For the test POSIX threads and default OS Processes were used. The main tasks to test were: one that was really intensive on the CPU, and another one intensive on the Disk I/O.

The process of testing was successful and it helped clear many of the hypothesis points and it showed better performance on processes than threads; nevertheless, the results as for which of these is better in general, and not only for the selected tasks, are inconclusive and will require more testing in different environments and using different tasks.

As a note for the reader, the conclusion was reached based only on the data from the experiment. Additional information such as security, availability, and more were not taken into consideration which resulted on not finding a general rule, but instead a specific one for the tasks tested.

Introduction

The use of threads and processes for the development of different applications is really common; nevertheless, since both of these serve really often the same purpose, which one to choose? The following report focuses on testing both of these in different environments to check which one performs better over most common tasks.

The results are expected to show which of these is better overall and therefore should be used over the other to increase the program's performance.

Theory Behind the Experiment

Threads and processes already have their own advantages such as how easy it is to share variables between threads since they all run within the same process, or how processes are more secure due to the privacy of the data between processes. However, there is not a clear proof as to which one is better to use in every case to make a program more efficient even if it increases overhead.

Before the experiment we expect threads to be easier to implement and work just as well as the processes but with less overhead, as for the processes prospect we expect processes to be a bit faster at memory swapping and doing calculations. These will be tested by performing the same tasks using only processes in one program and threads on the other.

Experimental Apparatus and Procedure

In order to compare the performance of threads and processes, 2 programs were created, one that is multithreaded with 8 POSIX threads and one with multiple processes, 8 children processes. Both of these ran the same tasks and it was checked the time each program took to complete the tasks as well as the use of CPU, RAM and Disk.

These programs were tested on 2 different environments. One desktop with 4 logical cores and a laptop with 8 logical cores, both having an HDD, 8GB RAM and operating on Fedora 22.

The two main tasks both programs had to execute were:

CPU Intensive.- Consists of calculating the divisors for a specific number. In order to keep the CPU working for a longer time, a “long long” number was used which resulted in a better assessment on how fast thread/processes are. On a later implementation this task would also allocate a small chunk of memory(400B) on each of its calls inside the loop to further compare efficiency. **6 workers ran this task.**

Disk I/O Intensive.- Consists of transferring a file over. It involves reading from a file, and writing it over somewhere else to test writing speeds as well as reading. **2 workers run this task.**

Each program kept a log file that will comprise of exact times for which each worker thread/process started and when they finished as well as a total execution time for each of them. With the examination of this data and some other such as CPU, Memory Usage we achieved a conclusion in the case of these tasks.

Results and Discussion

One of the first test on an early version of the CPU Intensive Task were it only calculated the divisors for a number, resulted on some mixed results which showed that on average processes will take a little longer than threads resulting on a program running time of 54.436 seconds for threads and 54.536 for processes (Fig. 1). When split into every worker process/thread, thread workers were on average faster than the processes for the CPU Intensive Task as well; nevertheless, the Disk I/O tasks showed a notable difference where processes took half the time compared to threads.

Program has started at Wed Jan 13 22:29:21 2016

Program has finished at Wed Jan 13 22:30:16 2016

Processes Program took a total time of 54536

Program has started at Wed Jan 13 22:27:49 2016

Program has finished at Wed Jan 13 22:28:44 2016

Threaded Program took a total time of 54436

Fig. 1

However since most programs need to also make heavy use of the memory, a loop was added for memory allocation for the CPU Intensive Task. After this change the results were more one sided showing that processes were in fact taking less time than threads in most or all of them. The results for the first test on the 4 logical cores computer was the following:

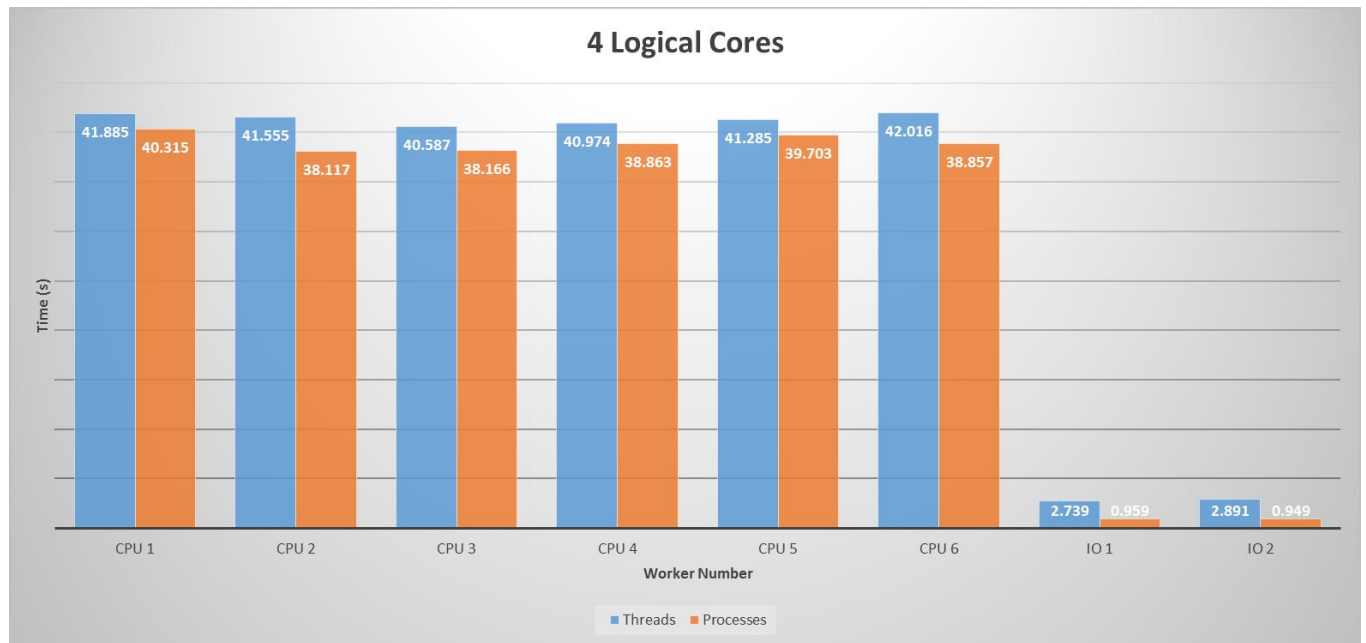


Fig. 2

As the graphic shows, Threads took on average about 3 more seconds to complete the CPU task compared to the processes. As for the Disk IO task, there is a clear difference on how processes are being complete as much as 3 times faster than threads. The results on this environment demonstrate that in fact processes are having a superior performance time-wise when compared to threads.

The results for the second test on the 8 logical cores computer was the following:

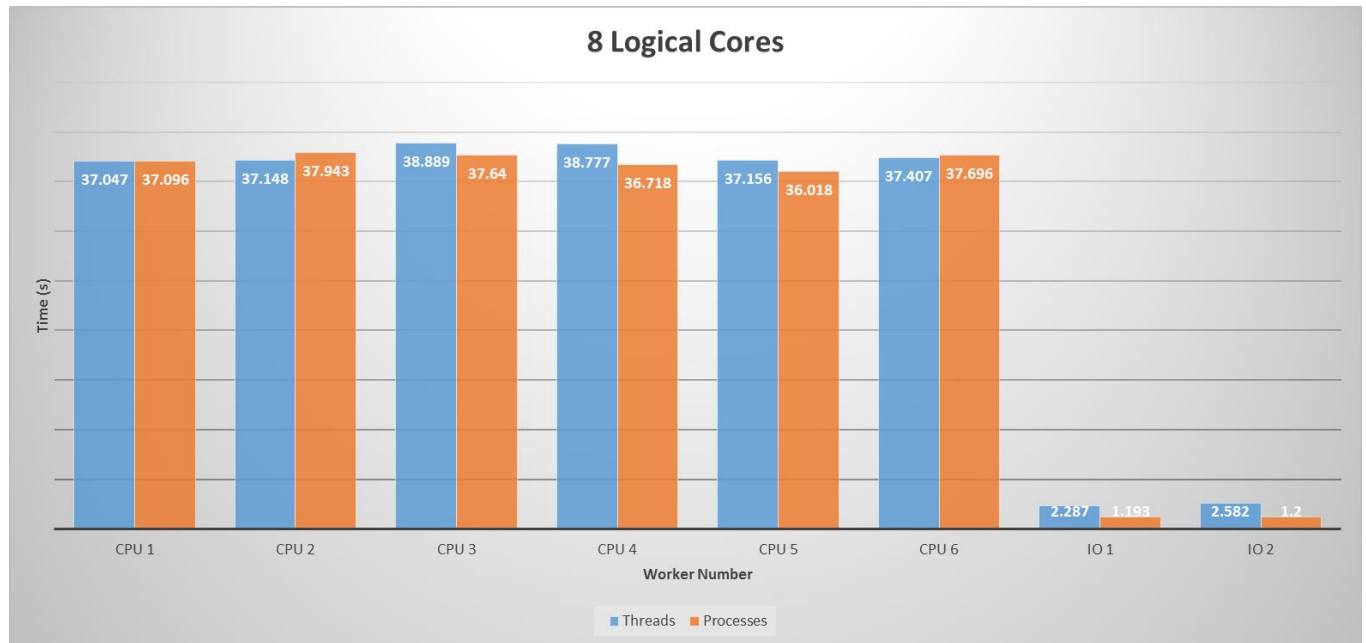


Fig 3

Just like on the other environment, Processes seem to be taking less time than Threads overall; however, it is not as one sided as when counting with only 4 cores. As for the Disk I/O, the difference keeps being clear, Threads seem to be lacking on the IO are and processes on this test were twice as fast. As an overall result on time efficiency this test showed that processes were still better.

Now we will proceed to discuss how Threads and Processes dealt with the use or the CPU on each environment as well of the memory use for each of them:

Threads

Running the multi-threaded program on the 4 cores machine resulted in having every core pinned at 100% during most of the execution time (Fig. 4) something expected in order to finish the task as fast as possible.

On the other hand, running the program on the 8 cores machine showed a different result on CPU Usage (Fig. 5). It started by using all of the 8 cores at 100% for a really short time. However, as soon as the **2 Disk IO Task finished** (around the 2 seconds mark) only 4 cores stayed pinned at 100% while the other 4 seem to oscillate between 20-100% sharing the workload. This resulted in a faster program execution and it was a bit unexpected since 6 cores pinned at 100% were expected instead of 4 sharing the workload.

As for the use of memory for the program the average memory use was 108 KB (Fig. 6) which seems pretty lightweight and it was pretty steady up until the program ended.

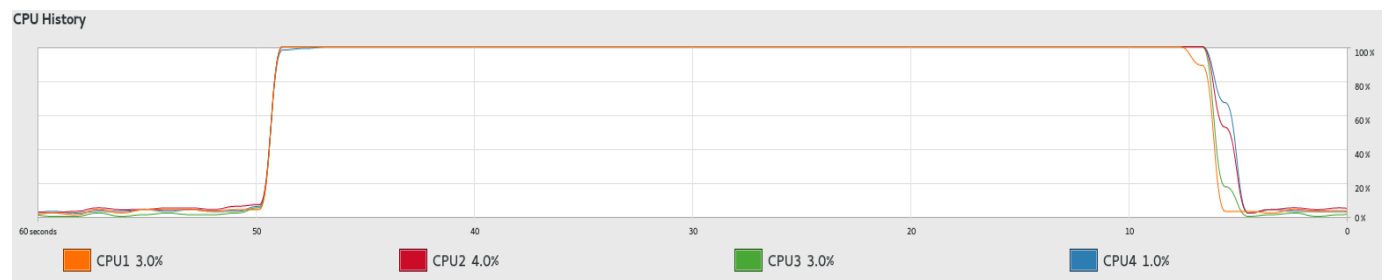


Fig. 4

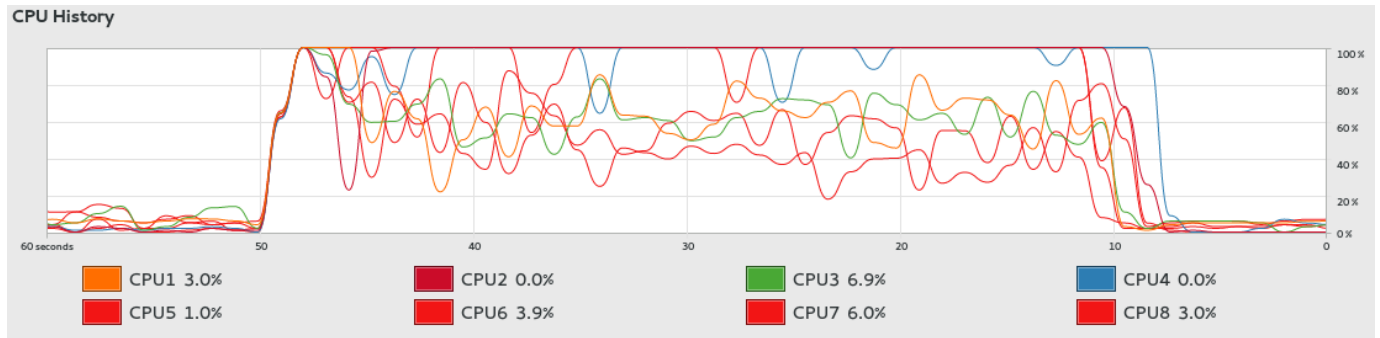


Fig. 5

Process Name	User	% CPU ▾	ID	Memory	Priority
threads	manuelg	74	2982	108.0 KiB	Normal

Fig 6.

Processes

Running the program with multiple processes on the 4 cores machine resulted in having every core pinned at 100% during most of the execution time (Fig. 7) . Just as the multi-threaded program and as expected

On the other hand, running the program on the 8 cores machine showed a different result on CPU Usage (Fig. 8). It started by using all of the 8 cores and then just like as for the multi-threaded program it pinned 4 cores and shared the workload between the other 4; however, one difference to note compared to the multi-threaded program is that since 2 of the 8 worker processes(the 2 running the **Disk IO Task**) finished really fast, CPU usage does not get to the point where all 8 cores are pinned at 100%.

As for the use of memory for this program the average memory use was 524 KB (Fig. 9) which is using about 5 times as much when compared to the multi-threaded program.

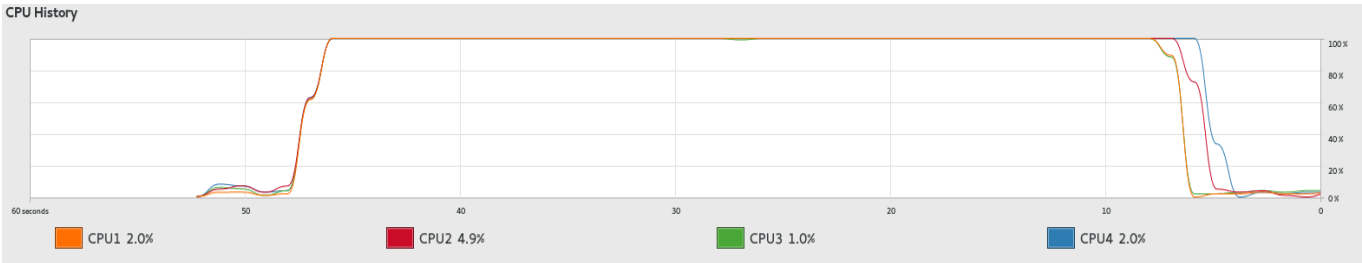


Fig. 7

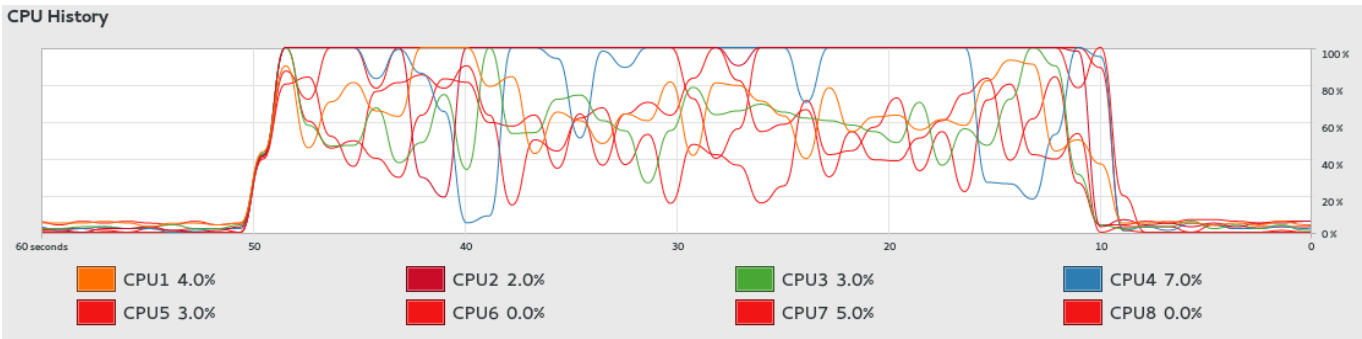


Fig. 8





Process Name	User	% CPU ▾	ID	Memory	Priority
 processes	manuelg	12	3090	84.0 KiB	Normal
 processes	manuelg	12	3091	88.0 KiB	Normal
 processes	manuelg	12	3092	88.0 KiB	Normal
 processes	manuelg	12	3093	88.0 KiB	Normal
 processes	manuelg	12	3094	88.0 KiB	Normal
 processes	manuelg	12	3095	88.0 KiB	Normal

Fig. 9

Conclusion

From this short experiment after reviewing the results, we can conclude that Processes seem to be more efficient for Disk I/O tasks and should be used in these types of tasks in order to critically increase the performance over threads. On the other hand, the results show that Processes are also slightly faster than threads for CPU Intensive Tasks and would be worth using as well in order to increase the program performance.

However, for the CPU Intensive task the difference is not as noticeable as it is for Disk I/O and even on some of the instances some worker Threads seemed to perform better. Furthermore, processes have more overhead than threads and on top of that they use more resources such as memory. Taking all of these into account Threads seem to be the best option when resources are limited.

To conclude, Processes are the best option overall for the discussed tasks since they are more efficient for the Disk I/O tasks and slightly faster on the CPU task. Regarding I/O tasks they are the best option; nevertheless, it is worth noticing that if there is a lack of resources, threads are a more suitable option and will perform really close to processes speeds for CPU tasks. Now the question is: Are the use of extra resources worth, in order to achieve a slightly better performance for CPU tasks?