**Computer Systems Technology**

British Columbia Institute of Technology

COMP 8005 - Assignment2- Report

Abt Hu

Feb 20, 2018

# Table of Contents

[Table of Contents 2](#_Toc413112767)

[1. Summary 3](#_Toc2086762790)

[2. Introduction 4](#_Toc180741042)

[3. Background 5](#_Toc227842858)

[4. Discussion 6](#_Toc635851960)

[5. Conclusion 11](#_Toc572568809)

# Summary

The purpose of this assignment was to analyze and compare the performance of multi-threaded and multi-processed. To do so, Python 2.7 was used in a Linux Fedora OS on Datacomm lab computers and in a Ubuntu OS on my laptop and a Raspbian OS on raspberry Pi 3.

To analyze and compare these two methods, a set of performance tests were developed to help show any differences between them. Testing programs were setup on various platforms I mentioned above in attempts to test and analyze performance, notably in regards to time consuming, memory&CPU usage.

My hypothesis was that the multi-threaded app would be outclassed by the my multi-processed app and the single-threaded app in turn would be outclassed by both of the multi-thread and multi-processed app. But our results proved more complicated. Time consuming were comparable between the different methods, but performance of my multi-threaded app was unable to defeat the multi-processed app. The multi-threaded app performance in some case is even worse than single-threaded one.

My discoveries were interesting. All apps slowed considerably as the calculation or IO workload went up. My multi-threaded approach show its advantage only in Internet requests works. While the multi-processed approach wins in both CPU-bound and IO-bound test, and in Internet request test both multi-threaded and multi-processed approach perform in the same level, very little difference.

# Introduction

The purpose of this assignment was to analyze and compare the performance between the multi-threaded, multi-processed method implementations. The tests were developed using Python 2.7 and were run on the Linux Fedora OS using Datacomm lab computers and Ubuntu OS using my laptop and Raspbain OS using raspberry Pi 3.

The goal of this assignment is to have a better understanding of the way different types of implementation perform on same platforms compared to each other.

# Background

To analyze and compare the performance of multi-thread, multi-processed approach, python programs were written for all three servers types as well as a common test using multi-processing and multi-threading.

The Multi-Threaded app creates a set of worker threads to manage each performance test. No data is need to receive and echoed back to main stream. When the work threads finish one test, then the main stream will print the time consuming message and generate a bunch of worker threads to perform the next test. These worker thread would experience three types of test. The first one is CPU-bound test, in this test, each worker thread would do 1,500,000 times adding calculation. The second test is IO-bound test. In this test, each thread would write millions of lines repeated words to a file, and then read this in to list. The last test is http request job, it’s just make the thread request a web page and return the web page.

The Multi-Processed app actually do the same thing as the Multi-Threaded app. The only different part is instead of using a bunch of working thread to do those test, this app generate a set of child processes. It also records those time consuming for these three type of tests.

# Discussion

My hypothesis was that the multi-threaded app would be outclassed by the my multi-processed app and the single-threaded app in turn would be outclassed by both of the multi-thread and multi-processed app.

To test my hypothesis, a formal series of tests and stress tests were run with in various platforms. The time consuming and memory&CPU usage were averaged and compared. I also collect the single-threaded approach for proving.

**CPU-bound test:**

Time-Consuming data

Env: Ubuntu OS: laptop

Unit: Seconds

Run for 1,500,000 times adding calculation each worker

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1st | 2nd | 3rd | ave |
| Single thread | 85.609539985 | 82.998448133 | 83.198224067 | 83.935404062 |
| Multi-thread | 86.755592823 | 86.794086933 | 88.320499897 | 87.290059884 |
| Multi-process | 46.361334085 | 49.290297985 | 47.057981968 | 47.569871346 |

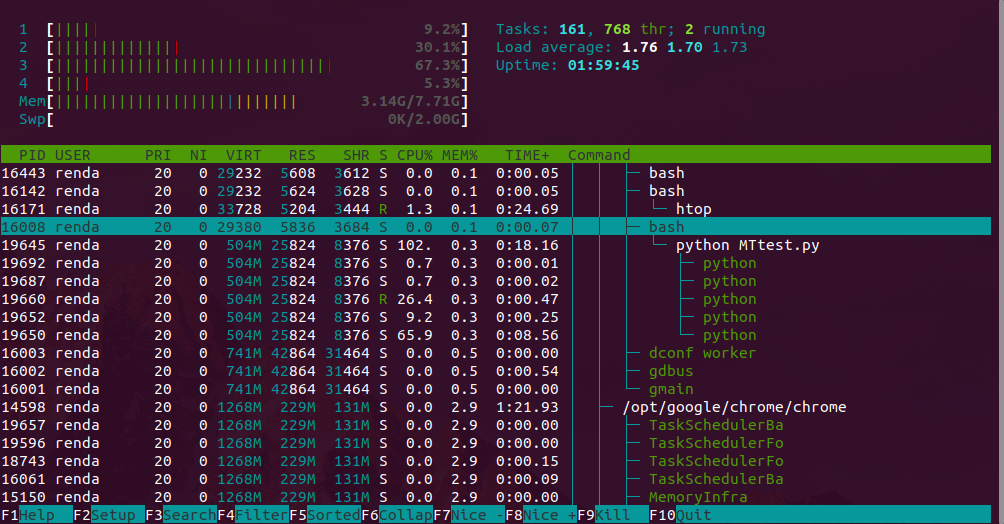
Env: Raspbian OS: Raspberry pi 3

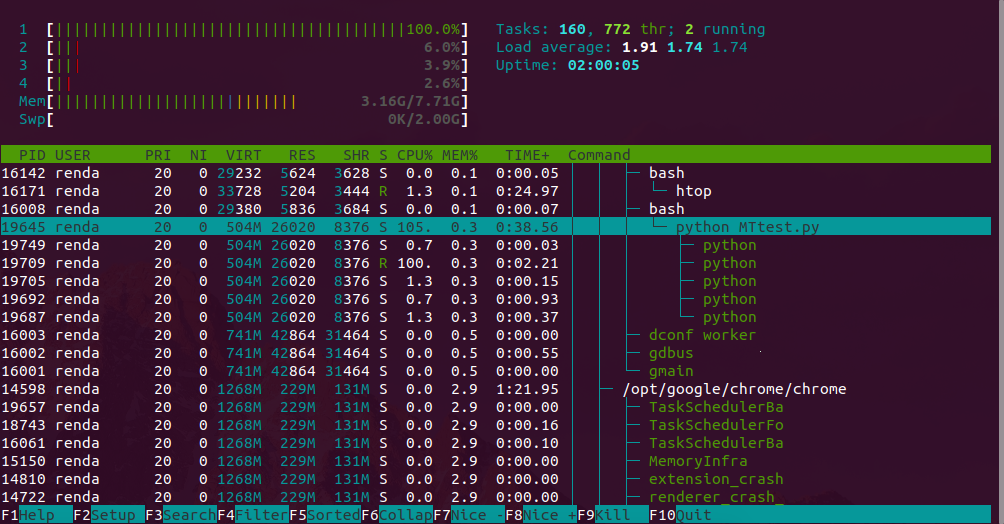
Unit: Seconds  
Run for 1,500,000 times adding calculation each worker

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1st | 2nd | 3rd | ave |
| Single thread | 1144.2190089 | 1274.0143559 | 1179.72878098 | 1199.32071526 |
| Multi-thread | 1140.2853100 | 1140.4750020 | 1145.62363505 | 1142.12798235 |
| Multi-process | 548.12708998 | 554.75838399 | 554.964472055 | 552.616648675 |

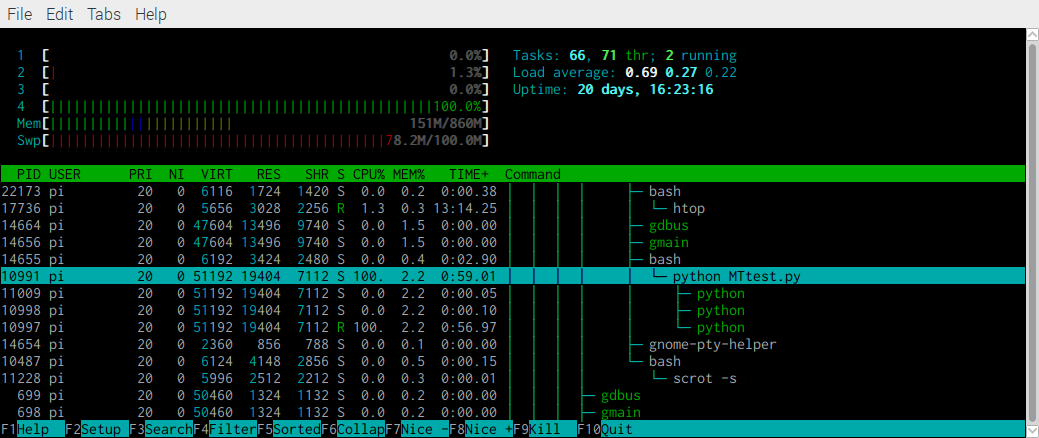
From time consuming result, it show that:

* Multi-threaded approach preformed as single-threaded approach in CPU-bound test. No advantages in comparison.
* Multi-processed approach is way faster than the rest two implementations, almost half of their time consuming.





Capture from Ubuntu



Capture from Raspbian

We can find that although in this test there are a set of threads, but only one of them is running at a time. In another word, it’s not real multi-thread should do.

**IO-bound test:**

Time-Consuming data

Env: Ubuntu OS: laptop

Unit: Seconds

Read and write for 2,500,000 lines each worker

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1st | 2nd | 3rd | ave |
| Single thread | 7.6832900047 | 7.3490431309 | 7.19875001908 | 7.41036105156 |
| Multi-thread | 27.402339935 | 26.879519939 | 26.7844421864 | 27.0221006868 |
| Multi-process | 3.4483058453 | 3.4645278454 | 3.49010896683 | 3.46764755251 |

Env: Raspbian OS: Raspberry pi 3

Unit: Seconds

Read and write for 1,500,000 lines each worker

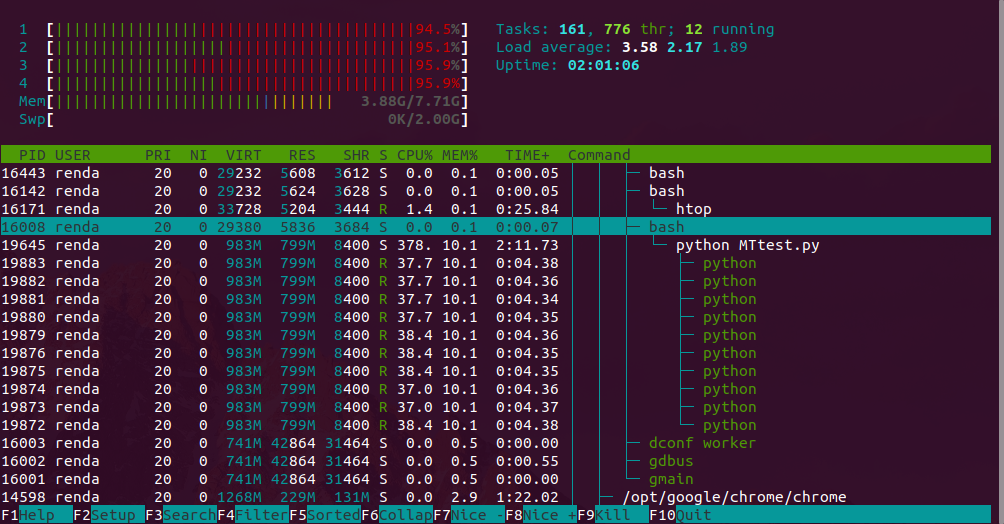
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1st | 2nd | 3rd | ave |
| Single thread | 40.157208919 | 39.686310053 | 41.1200339793 | 40.3211843171 |
| Multi-thread | 83.371158123 | 82.913375139 | 84.3446321487 | 83.5430551369 |
| Multi-process | 15.786888123 | 16.368587017 | 16.0874328614 | 16.0809693338 |

From time consuming result, it show that:

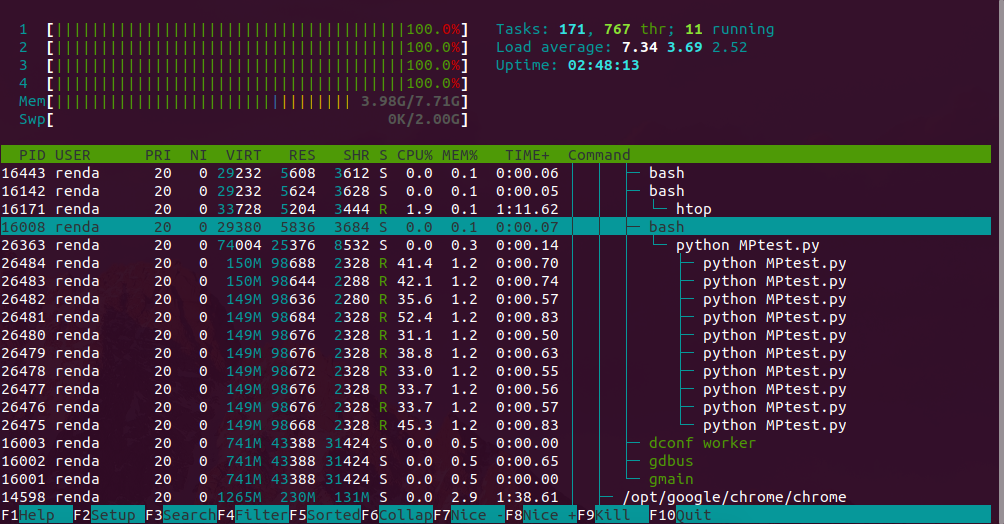
* Multi-threaded approach preformed even worse than single-threaded approach in IO-bound test.
* Multi-processed approach is way faster than the rest two implementations, almost 5 times than multi-threaded implementation.

Memory-Consuming data

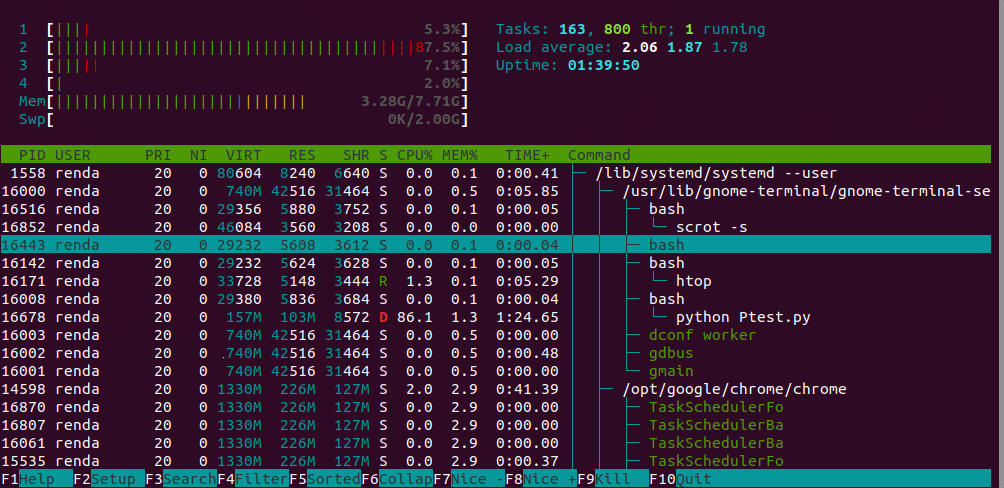
Env: Ubuntu OS: laptop



Data from multi-threaded implementation



Data from multi-processed implementation



Data from single-threaded implementation

|  |  |
| --- | --- |
|  | Memory Usage(VIRT) |
| Single thread | 157M |
| Multi-thread | 983M |
| Multi-process | 1564M |

From memory usage result, it show that:

* Multi-threaded approach memory usage is way better than Multi-processed implementation, while can’t compete with single-threaded implementation.
* Multi-processed approach use a lot of memory for those job, so if we are short of memory maybe this implementation is not good for it.

**HTTP request test:**

Time-Consuming data

Env: Ubuntu OS: laptop

Unit: Seconds

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1st | 2nd | 3rd | ave |
| Single thread | 21.662107945 | 24.353891850 | 22.4338281155 | 22.8166093035 |
| Multi-thread | 2.3229291439 | 2.2970578671 | 2.61981606482 | 2.41326769194 |
| Multi-process | 2.3581211566 | 2.2979779244 | 2.70154881477 | 2.45254929859 |

Env: Raspbian OS: Raspberry pi 3

Unit: Seconds

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1st | 2nd | 3rd | ave |
| Single thread | 28.331081152 | 29.605278015 | 28.2196531295 | 28.7186707655 |
| Multi-thread | 3.7372238636 | 3.8312768936 | 4.41367101668 | 3.99405725796 |
| Multi-process | 4.1208260059 | 4.0606040955 | 3.18546199799 | 3.78896403313 |

From time consuming result, it show that:

* Multi-threaded approach preformed is way better than single-threaded approach in this test. And it perform as good as multi-processed one. Considering its memory saving feature, maybe it’s good for this kind of job.
* Multi-processed approach performance is almost the same as multi-threaded implementation. While due to its high memory usage, maybe it is not good enough for http request job.

# Conclusion

From the data collected, it can be seen that Multi-processed approach is the superior choice for managing high pressure CPU-bound job and IO-bound job. The time consuming for both approach were similar for web request job test and the Multi-Threaded approach can save some memory.

The Multi-Threaded approach performed not good, even compare with the single-threaded approach in CPU-bound test and IO-bound test, Multi-Threaded approach have no much advantage. Consider about the Cpython have GIL(Global Interpreter Lock), therefore, the thread is concurrent but not parallel. So when we are running CPU-bound job, the Multi-Threaded approach perform worse than single-threaded. While in web request job, the Multi-Threaded approach is more comfortable than single-threaded.

The Multi-Processed approach perform much better in both CPU-bound and IO-bound tests. And in web request job it still good. The only problem is this approach occupy much more memory resources than other approaches. So we may choose Multi-Threaded approach more in web request task.