|  |
| --- |
| BCIT BTech: Network security & Administration – SET 5d |
| COMP 8005 Assignment 01 |
| Measuring the performance and efficiency of using multiple processes and threads. |
|  |
| **Thilina Ratnayake – A00802338** |
| **1/26/2015** |

|  |
| --- |
| An experiment to observe the difference in performance between multi-threading versus multi-processing. |

**Table Of Contents:**

|  |  |
| --- | --- |
| Page | Item |
| 2 | Objective |
| 2 | **REPORT & EXPERIMENT DESIGN** |
| 2 | Controls |
| 3 | Results |
| 6 | Observations and Remarks |
| 6 | Conclusion |
|  | **PROGRAM DESIGN** |
| 7 | Finite State Machines |
| 8 | Program Components |
| 9 | Program Structure |
| 10 | Psuedo-Code: Processes |
| 11 | Psuedo-Code: Threads |
| 12 | Execution Instructions |
| 13 | **TESTING** |
| 14 | **Appendix A: Results in tabular form.** |

**Objective:**

Use multiple processes and threads on the Linux operating system and measure the differences of performance (if any) between each mechanism.

**Experiment Design**

1. The experiment will measure the performance of each subject (processes or threads) based on time taken to finish a task.
2. The language that will be used is C, and will be written for and tested on the Linux OS.
3. The experiment involves both Mathematical Computations and file I/O.  
   1. Mathematical computations: A string (“HASHSTRINGX” where X will be the iteration number) will be hashed using the SHA1 hashing algorithm a specified number of times.
   2. File I/O: The plaintext and hashed text will be printed onto text files.
4. Both subjects will be tested on hashing 1000,10000,100000,1000000, and 10,000,000 times.

**Controls:**

1. The experiment will be run 3 times for each subject, and completion times will be averaged.
2. The experiment will be run on the Fedora 21 distro.
3. The experiment will be run on two sets of hardware/environments.  
     
   Specifications:

|  |  |
| --- | --- |
| **Environment 1 -** BCIT Lab | **Environment 2 -**Personal PC with Linux VM |
| * Processor Cores: 4 * Processor: Intel Core i5-2400k CPU @ 3.10GHz -64bit * RAM:8GB | * Processor Cores: 4 * Processor: Intel Core i5-4670K CPU @ 3.40 GHz -64bit * Processor Speed: * RAM:1GB |

**Results:**

BCIT Lab:

**Fig 1A: Chart of performance times over amount of iterations hashed in Environment 1(BCIT Lab)**

Linux VM:

**Fig 1C: Chart of performance times over amount of iterations hashed in Environment 2 (Linux**

**VM)**

Final Comparison

**Fig 1E:** Chart comparing results from both environments.

**Observations & Remarks:**

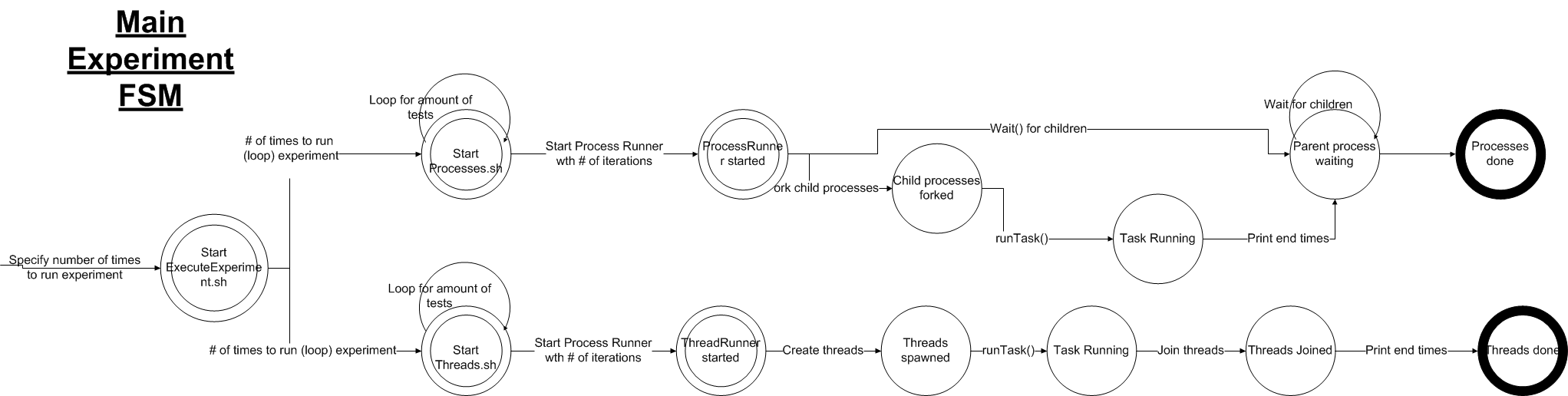
1. The performance between the two mechanisms are the same when dealing with a task of requiring lower processing requirements, as demonstrated by their performance from iterations 1000 to 100,000.
2. As compared to my colleague’s experiments which utilized prime number decomposition as it’s task instead of hashing, their results indicated that the difference in performance between the two mechanisms was marginal through-out the experiment, whereas in this experiment there is a drastic change in performance.
3. In trying to understand why multi-threading took so much time, the sched\_yield() call and mutex\_locks were taken out as they do not utilize any of the same resources, thus avoiding any race conditions. This change in the program only reduced ~30 seconds off the total time.

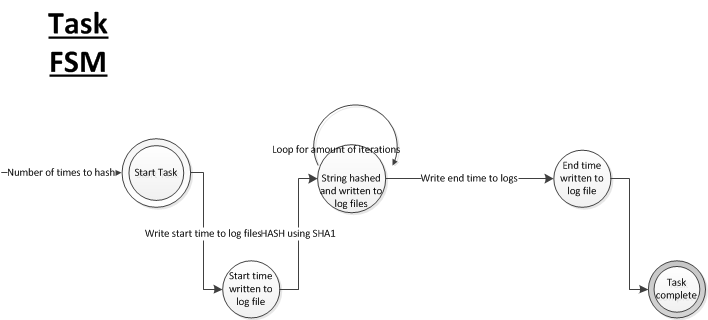
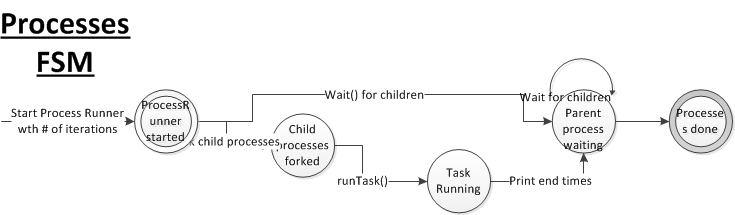
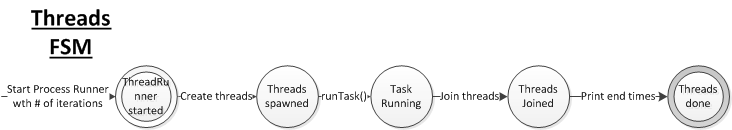
**Conclusion**

The conclusion that can be reached from this experiment is that the efficiency of utilizing a multi-threading mechanism decreases with the rise of processing requirements for a specific task. When compared to a multi-processing mechanism, which is approximately 291% more efficient than multi-threading at a task of 10,000,000 iterations.

**DESIGN**

**Finite State Diagrams**

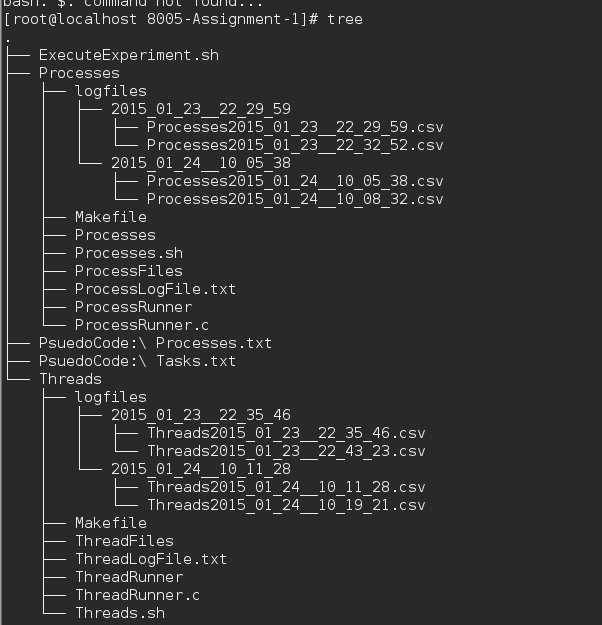
*(These diagrams can be found in /FSMs for closer inspection)* ****



**Program Components:**

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Functions | Inputs | Outputs |
| ExecuteExperiment.sh | Script that runs Processes.sh and Threads.sh one after the other. | Number of times to run tests | -- |
| Processes.sh | Script that runs ProcessRunner.c in a loop with desired amount of hash iterations. | Number of times to run tests (ProcessRunner) | ./logfiles/DATETIME/DATETIME.csv |
| Threads.sh | Script that runs ThreadRunner.c in a loop with desired amount of hash iterations. | Number of times to run tests(ThreadRunner) | ./logfiles/DATETIME/DATETIME.csv |
| ProcessRunner.c | Spawns 5 child processes and executes a task that hashes a string for the specified amount of iterations and outputs to logfiles. | Number of iterations. | ./ProcessFiles/ProcessTaskOutputFileX.txt (where X = Process number)  *Note:*  When run with the script, all output files will be deleted as part of tear down procedure to save disk space. |
| ThreadRunner.c | Spawns 5 child threads and executes a task that hashes a string for the specified amount of iterations and | Number of iteartions. | ./ThreadFiles/ThreadTaskOutputFileThreadX.txt (where X = Thread Number)  *Note:*  When run with the script, all output files will be deleted as part of tear down procedure to save disk space. |

**Program Structure:**

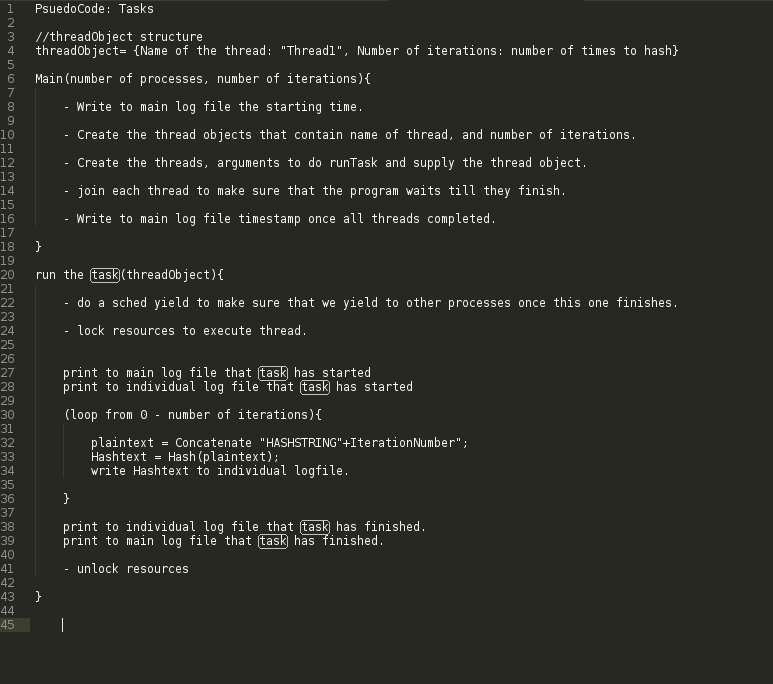
****

**Psuedo-Code**

Processes:

****

Threads:

****

**Program Requirements:**

1. openssl-devel-1:1.0.1j-1.fc21.x86\_64

**Execution Instructions**

To execute the experiment as it was done for gathering data.

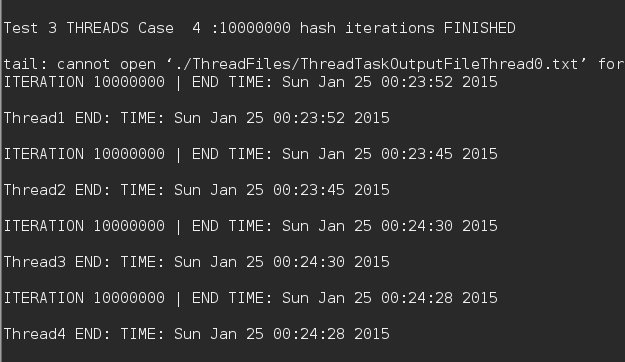
1. Install required libraries
   1. *Yum install openssl-devel*
   2. *Yum install openssl-libs-1.0.1j-1.fc21.x86\_64 zlib-1.2.8-7.fc21.x86\_64*
2. Compile source code.
   1. *CD ./Processes*
   2. *Make ProcessRunner*
   3. *CD ../Threads*
   4. *Make ThreadRunner*
3. Within main directory, execute
   1. *SH ExecuteExperiment.sh 4*

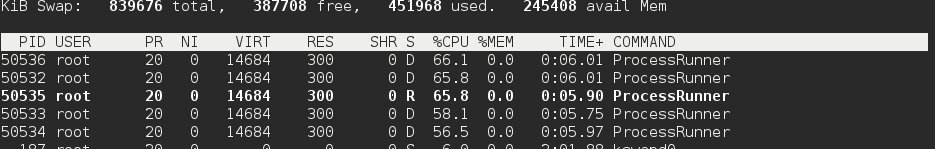
To only run the Thread or ProcessRunner

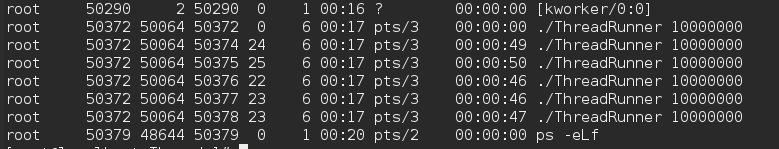
1. Install required libraries
   1. *Yum install openssl-devel*
   2. *Yum install openssl-libs-1.0.1j-1.fc21.x86\_64 zlib-1.2.8-7.fc21.x86\_64*
2. Compile with following flags
   1. –lcrypto -pthread

**Testing**

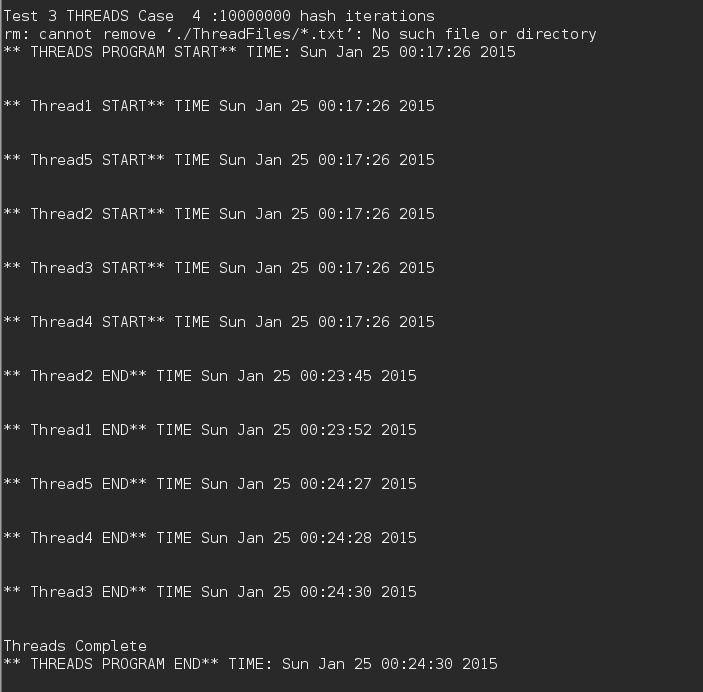
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Name | Resources | Expected | Actual | Fig |
| 1 | Hash specified amount of times | ProcessRunner.c,ThreadRunner.c | The end of the log file to contain the last hash number which should be the same as the iteration number | Same as expected | 1.1 |
| 2 | Generate unique hashes. | ThreadOutputFileX.txt, ProcessOutputFIleX.txt | The hashes should all be different from each other. | As expected | 4.1 |
| 3 | Spawn 5 Threads/Processes | ThreadRunner.c,ProcessRunner.c | 1 parent thread and 5 child threads. | Same as expected. | 2.1A,2.1B |
| 4 | Wait for child processes / other threads. | ProcessRunner.c,ThreadRunner.c | Program will not finish (print out end time) until tasks and processes have all finished writing to file. | Same as expected. | 3.1A,3.1B |

**Fig 1.1 Hash specified amount of times.** (“Iteration 10000000 | END TIME …)” is the output of a “tail –n4” command reading from a log file. This proves that 1000000 iterations were written.

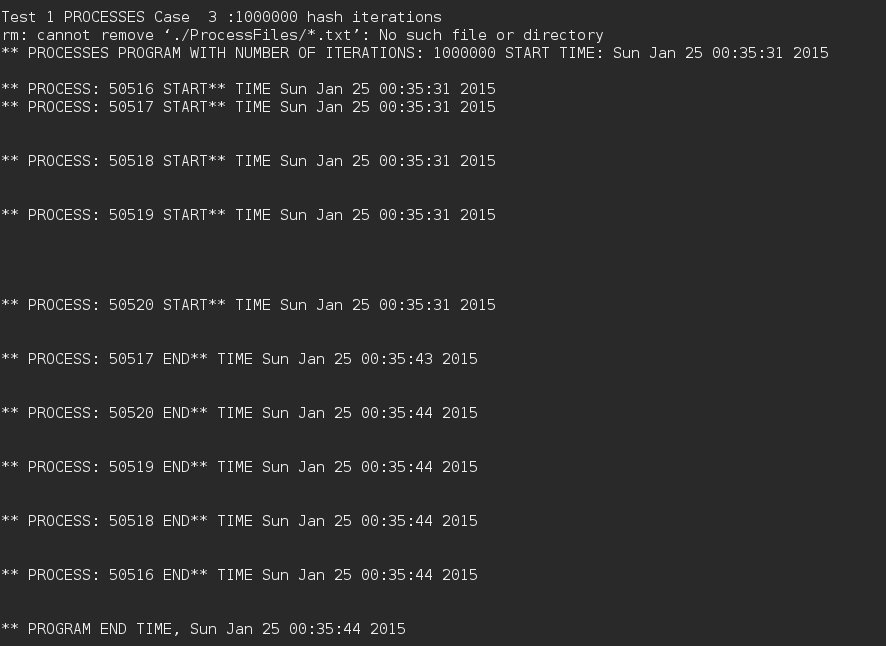
**Fix 2.1A 5 child processes running**



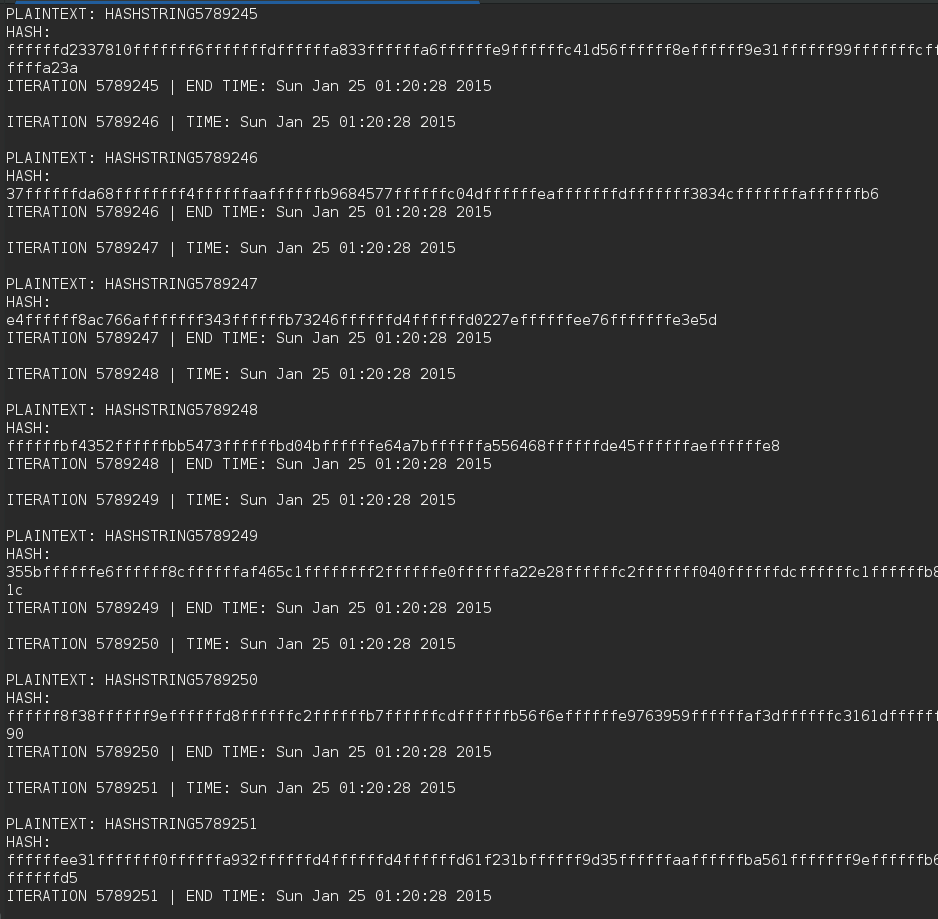
**Fig 2.1B 5 child threads running.**



**Fig 3.1A Waiting for child threads to finish before exiting.**



**Fig 3.1B Waiting for child processes to finish before exiting.**



**Fig 4.1 Generate unique hashes**

**Annex A: Experiment Data**

Environment 1 (BCIT Lab) Data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Iterations | | | | |
| Processes | 1000 | 10000 | 100000 | 1000000 | 10000000 |
| Average Time | 0:00:00 | 0:00:00 | 0:00:02 | 0:00:13 | 0:02:12 |
| Test 1 | 0:00:00 | 0:00:01 | 0:00:01 | 0:00:14 | 0:02:11 |
| Test 2 | 0:00:00 | 0:00:00 | 0:00:02 | 0:00:13 | 0:02:10 |
| Test 3 | 0:00:00 | 0:00:00 | 0:00:02 | 0:00:13 | 0:02:14 |
|  |  |  |  |  |  |
|  | Iterations | | | | |
| Threads | 1000 | 10000 | 100000 | 1000000 | 10000000 |
| Average Time | 0:00:00 | 0:00:00 | 0:00:04 | 0:00:37 | 0:06:21 |
| Test 1 | 0:00:00 | 0:00:00 | 0:00:04 | 0:00:38 | 0:06:26 |
| Test 2 | 0:00:00 | 0:00:00 | 0:00:04 | 0:00:37 | 0:06:21 |
| Test 3 | 0:00:00 | 0:00:00 | 0:00:04 | 0:00:37 | 0:06:17 |
| Graph Data |  |  |  |  |  |
| Graph Data | Iterations | | | | |
| Average time over 3 tests | 1000 | 10000 | 100000 | 1000000 | 10000000 |
| Processes | 0:00:00 | 0:00:00 | 0:00:02 | 0:00:13 | 0:02:12 |
| Threads | 0:00:00 | 0:00:00 | 0:00:04 | 0:00:37 | 0:06:21 |

**Fig 1B: Data of performance times over amount of iterations hashed in Environment 1 (BCIT Lab)**

Environment 1 (Linux VM) Data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Iterations | | | | |
| Processes | 1000 | 10000 | 100000 | 1000000 | 10000000 |
| Average Time | 0:00:00 | 0:00:00 | 0:00:02 | 0:00:13 | 0:02:18 |
| Test 1 | 0:00:00 | 0:00:00 | 0:00:02 | 0:00:13 | 0:02:20 |
| Test 2 | 0:00:00 | 0:00:00 | 0:00:01 | 0:00:13 | 0:02:17 |
| Test 3 | 0:00:00 | 0:00:00 | 0:00:02 | 0:00:13 | 0:02:18 |
|  |  |  |  |  |  |
|  | Iterations | | | | |
| Threads | 1000 | 10000 | 100000 | 1000000 | 10000000 |
| Average Time | 0:00:00 | 0:00:01 | 0:00:04 | 0:00:41 | 0:06:42 |
| Test 1 | 0:00:01 | 0:00:00 | 0:00:04 | 0:00:41 | 0:06:40 |
| Test 2 | 0:00:00 | 0:00:01 | 0:00:04 | 0:00:44 | 0:06:57 |
| Test 3 | 0:00:00 | 0:00:01 | 0:00:03 | 0:00:39 | 0:06:30 |

Graph Data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Graph Data | Iterations | | | | |
| Average time over 3 tests | 1000 | 10000 | 100000 | 1000000 | 10000000 |
| Processes | 0:00:00 | 0:00:00 | 0:00:02 | 0:00:13 | 0:02:18 |
| Threads | 0:00:00 | 0:00:01 | 0:00:04 | 0:00:41 | 0:06:42 |

**Fig 1D: Data of performance times over amount of iterations hashed in Environment 2 (Linux VM)**