Assignment 2: Processes vs Threads (Round 0)

Jesse Gatling - net ID: jag548
Andre Pereira - net ID: andreper

Testing

Test 1 - Number of characters = number of partitions -> end result = expected result.

Test 2 - Number of characters = 1 and number of partitions >= number of characters -> end result = expected result.*

Test 3 - Missing args/Erroneous args -> Error Reported and no threads or processes created.

Test 4 - Ten rounds of analysis of runtimes of same file with same number of partitions (10 partitions here):

Trial One:
Processes: real runtime -> .074 seconds.
Threads: real runtime -> .035 seconds.

Trial Two:
Processes: real runtime -> .073 seconds.
Threads: real runtime -> .062 seconds.

Trial Three:
Processes: real runtime -> .072 seconds.
Threads: real runtime -> .028 seconds.

Trial Four:
Processes: real runtime -> .073 seconds.
Threads: real runtime -> .074 seconds.

Trial Five:
Processes: real runtime -> .073 seconds.
Threads: real runtime -> .035 seconds.

Trial Six:
Processes: real runtime -> .066 seconds.
Threads: real runtime -> .060 seconds.

Trial Seven:
Processes: real runtime -> .101 seconds.
Threads: real runtime -> .053 seconds.

Trial Eight:
Processes: real runtime -> .075 seconds.
Threads: real runtime -> .060 seconds.

Trial Nine:
Processes: real runtime -> .072 seconds.
Threads: real runtime -> .054 seconds.

```
          Trial Ten:
               Processes: real runtime -> .075 seconds.
               Threads: real runtime -> .028 seconds.
```

Test 5 - Ten rounds of analysis of runtimes of same file with different number of partitions:

```
          Trial One: 3 partitions
               Processes: real runtime -> .119 seconds.
               Threads: real runtime -> .131 seconds.
          Trial Two: 5 partitions
               Processes: real runtime -> .175 seconds.
               Threads: real runtime -> .121 seconds.
          Trial Three: 7 partitions
               Processes: real runtime -> .094 seconds.
               Threads: real runtime -> .127 seconds.
          Trial Four: 9 partitions
               Processes: real runtime -> .174 seconds.
               Threads: real runtime -> .139 seconds.
          Trial Five: 11 partitions
               Processes: real runtime -> .175 seconds.
               Threads: real runtime -> .137 seconds.
          Trial Six: 13 partitions
               Processes: real runtime -> .066 seconds.
               Threads: real runtime -> .118 seconds.


          Trial Seven: 17 partitions
               Processes: real runtime -> .065 seconds.
               Threads: real runtime -> .031 seconds.
          Trial Eight: 19 partitions
               Processes: real runtime -> .086 seconds.
               Threads: real runtime -> .135 seconds.
          Trial Nine: 23 partitions
               Processes: real runtime -> .152 seconds.
               Threads: real runtime -> .038 seconds.
          Trial Ten: 29 partitions
               Processes: real runtime -> .107 seconds.
               Threads: real runtime -> .048 seconds.
```

Test 6 - Partitioning a very large file at our recommended partitioning number for large files with threads:

OS should complain when doing so and we received the complaint--a segmentation fault.

Learned that for files of size 100.0kB, the maximum number of partitions appears to be 71.

Test 7 - Partitioning a very large file at our recommended partitioning number for large files with processes:

Program output the correct number of files and correct compression. Processes are BETTER for large files with many multiple partitions.

Conclusions

Based on our observations, we have concluded that for text files of less than 10mB, it would be in the programmer’s best interest to compress the file using threads rather than processes. The clocked time for threads in some cases was faster than that of processes. In general, however, the clock speed of processes *appeared* to be consistent here.

For text files of a very large size, it is in the programmer’s best interest to run this program with processes. The user will be given much more access to memory this way and he or she will then be able to easily compress large amounts of data without worrying about matters such as heap memory.

In a nutshell, if the user plans to use a lot of the heap memory, the user would be best advised to use processes over threads. Otherwise, threads appears to be the better contender here. Speed over consistent performance (from processes) is the deciding factor for choosing threads.