Portfolio Milestone 3

Brady Chin

Colorado State University Global

CSC507-2: Foundations of Operating Systems

Dr. Joseph Issa

February 2nd, 2025

Portfolio Milestone

Table 1 shows a summary of the method used and the time taken to execute.

Table 1: Method used and execution times

	Description	Execution Time (s)
numbers.sh	Shell script using a for loop	6
Basic Python script	Python script generating numbers using a single process	0.45
Multithreading Python Script	Used the *threading* module to generate numbers in parallel threads	0.42
Multiprocessing Python Script	Creates multiple processes then merges files using *shutil*	0.24

Why Each Method Was Chosen

The basic Python process was chosen to examine how Python would do using a very straight-forward single-threaded approach. Generally, Python is preferred for readability and flexibility, so it needed to be checked how well this would handle without the use of other optimization techniques. This is the most direct comparison to the numbers.sh script.

The multithreading approach has been used to test if the threading capability of Python may enhance performance due to the simultaneous execution of several threads. While generally threading has been suitable for I/O-bound tasks, it is expected that this method will handle multiple tasks concurrently; hence, there was some speeding up.

At last, it selected the most optimized multiprocessing to consider the advantages of parallelization by performing more than one process at once, with full use of multiple CPU cores. Since Python's multiprocessing library bypasses the GIL, it was expected to have the most improvement in performance, especially on tasks which are easy to parallelize-like this one.

Findings

The most time-consuming program was a shell script named numbers.sh using a for loop to generate random numbers, finishing its execution after 6 seconds. This turns out to be surprising since most of the readers may expect such a small task from a script to be rather efficient. Nevertheless, the simplicity of the process probably caused several delays due to its dependency on the shell environment, which in this case could not be prepared for this job as much as the other two methods.

The following simple Python script single-process-generated the numbers was finished in 0.45 seconds. This was substantially faster than the shell script but it still told us that the single-threaded performance of Python was worse than the shell script due to the overhead in starting up Python and dealing with file I/O operations.

The multithreading Python script, which employed Python's threading module to execute parallel threads of number generation, took 0.42 seconds to finish. This method indeed showed a slight improvement over the basic Python script by allowing it to execute several threads concurrently. However, this did not really perform much better, since Python's GIL prevented true parallelism, especially when the task is not CPU-bound.

The multiprocessing Python script had the best performance; it was able to accomplish the task in 0.24 seconds. Here, independent processes to generate numbers were spawned and used shutil to merge the files, hence taking advantage of many CPU cores available in the system. Unlike multithreading, multiprocessing avoids the GIL altogether, enabling true parallel execution and offering massive performance gains for this task.

Doubling CPU Power

In the event that the CPU power for a system were to be doubled, the effect would vary in the methods applied. In the case of the bash script, this would hardly affect it because the process is I/O-bound rather than CPU-bound and throughput excludes time for I/O or running other programs (geeksforgeeks.org, 2024, December 28)This would mean that the process

would still struggle with the writing speed of a disk rather than the available CPU resources, meaning that the effect would be small.

For the plain Python script, nothing would probably be improved, that is, this is a single-process and I/O-bound job; adding more CPU would not alleviate the I/O bottleneck for writing in a file. It seems then, for this type of process, scaling by using more central processing units isn't going to have much of an effect with respect to the execution speed.

However, the largest increase in CPU powers would be exploited by multiprocessing, since it involves several processes executing parallel, so that doubling the number of CPU cores available would have them execute concomitantly, hence up to 2x speed-up possible. This is because one can split a task across more CPU cores, hence maximizing the utilization of available hardware resources (Sirois, S., 2024, August 2).

References

geeksforgeeks.org, (2024, December 28) *Performance of Computer in Computer Organization.* https://www.geeksforgeeks.org/computer-organization-performance-of-computer/

Sirois, S., (2024, August 2) *Understanding CPU Speed: What Makes a Good Processor for Your Computer.* HP. https://www.hp.com/us-en/shop/tech-takes/what-is-processor-speed