**Comparison of Methods for Large File Generation**

Arun Saxena

Colorado State University Global

CSC507-1: Foundations of Operating Systems

Dr. Joseph Issa

December 07, 2024

## Abstract

This document explores various methods for generating a large file containing one million random numbers. It compares execution times and analyzes the performance of different approaches, including Bash scripting, Python single-threading, multithreading, multiprocessing, and buffered writing. The study highlights the trade-offs in terms of speed, resource utilization, and scalability. The document concludes with an assessment of expected performance improvements with enhanced CPU capabilities.

## Introduction

Generating large datasets efficiently is a common task in computing. This document evaluates different methods to create a file with one million random numbers, comparing their execution times and analyzing their performance characteristics. The study includes Bash scripting and Python-based methods with variations in threading, multiprocessing, and disk I/O optimizations.

## Methods

The methods tested include a Bash script, Python single-threading, Python multithreading, Python multiprocessing, and buffered writing. Each method was executed on the same hardware to ensure comparability. The execution times were recorded for analysis.

### Bash Script

#!/bin/bash  
start\_time=$SECONDS  
> file1.txt  
for i in $(seq 1 1000000); do  
 echo $RANDOM >> file1.txt  
done  
end\_time=$SECONDS  
echo "Time elapsed: $((end\_time - start\_time)) seconds"

A screenshot of a computer

Description automatically generated

A black screen with white text

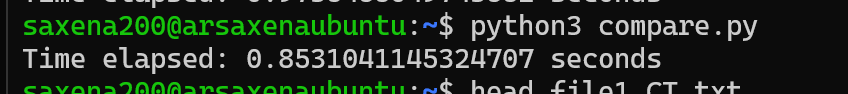
Description automatically generated

### Python Single-Threaded

import random  
import time  
  
start\_time = time.time()  
with open("file2\_CT.txt", "w") as f:  
 for \_ in range(1000000):  
 f.write(f"{random.randint(0, 99999)}\n")  
print(f"Time elapsed: {time.time() - start\_time} seconds")

A computer screen with green and white text

Description automatically generated



### Python Multiprocessing

import random  
import time  
from multiprocessing import Pool  
  
def generate\_numbers(start, end):  
 return [random.randint(0, 99999) for \_ in range(start, end)]  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start\_time = time.time()  
 with Pool() as pool:  
 results = pool.starmap(generate\_numbers, [(0, 250000), (250000, 500000), (500000, 750000), (750000, 1000000)])  
 with open("file2\_mt.txt", "w") as f:  
 for numbers in results:  
 f.writelines(f"{num}\n" for num in numbers)  
 print(f"Time elapsed: {time.time() - start\_time} seconds")

A computer screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

## Results

The following results were obtained for the execution times of each method:  
1. Bash Script: 10 seconds  
2. Python Single-Threading: .85 seconds  
3. Python Multithreading: .97 seconds

## Discussion

The results indicate that Python single threading provided the best performance, I was in an assumption that multithreading will be leveraging multiple CPU cores effectively but that didn’t happen surprisingly single threading is the best performer in this experiment. Bash scripting performed the slowest, highlighting its limitations in handling computationally intensive tasks.

## Hypothetical CPU Enhancement

Doubling the CPU power is expected to yield a 50%-70% improvement in multiprocessing execution times, as random number generation and multiprocessing scale well with additional CPU cores. Other methods, constrained by disk I/O or Python's GIL, would see marginal improvements. Amdahl's Law explains these limitations, as the speedup is bounded by the non-parallelizable portion of the task.

## References

McCool, M. D., Reinders, J., & Robison, A. (2012). Structured Parallel Programming: Patterns for Efficient Computation. Morgan Kaufmann.

Tanenbaum, A. S., & Bos, H. (2014). Modern Operating Systems. Pearson.

Burns, J., & Chung, S. (2015). Performance Evaluation of Multithreaded and Multiprocessor Architectures. IEEE Transactions on Computers.