**Portfolio Milestone 4: Option #2: Using Python Programming language**

Chioma Chance

Colorado State University Global

CSC507: Ethical Leadership in Software Development

Dr. L

01/12/2025

**Introduction**

Memory and file processing play a crucial role in computer systems, especially in optimizing the performance of data-heavy operations. This assignment explores different methods of processing a file (file1.txt) containing one million random numbers to double their values and write the results to a new file (newfile1.txt). The methods include a **Bash script** and three approaches in **Python**, each with its own strengths and weaknesses. By comparing their execution times and behaviors, we can better understand which method is most efficient under various conditions.

**Methods and Results**

**Bash Script (double\_numbers.sh)**

The Bash script uses a while loop to read each number from the input file (file1.txt), doubles the value using let, and appends the result to newfile1.txt. The time taken is measured using the SECONDS variable.

* **Execution Time**: 27 seconds (as shown in Screenshot 1).

**Observation**:  
The Bash script performed adequately but took longer due to the overhead of reading and writing each line individually. Bash is less optimized for handling large-scale file operations compared to Python.

**Python Methods**

**Method 1: Read Entire File into Memory**  
This method reads the entire contents of file1.txt into memory at once, processes all lines to double the values, and writes the results to newfile1.txt.

* **Execution Time**: 0.44 seconds (Screenshot 2).

**Observation**:  
This method is efficient for systems with sufficient memory, as it avoids the overhead of reading and writing line-by-line. However, it could face memory issues with larger files.

**Method 2: Read One Line at a Time**  
In this method, each line is read individually, processed, and written to newfile1.txt in the same loop.

* **Execution Time**: 0.50 seconds (Screenshot 2).

**Observation**:  
This method is more memory-efficient as it processes one line at a time, but it is slightly slower due to frequent file I/O operations.

**Method 3: Split File into Two Parts**  
The file is divided into two equal parts using the split command. Each part is processed separately, and the results are appended to newfile1.txt.

* **Execution Time**: 0.51 seconds (Screenshot 2).

**Observation**:  
Splitting the file distributes the workload, making it manageable for systems with limited memory. However, the extra step of combining the results adds to the overall execution time.

**Comparison of Methods**

| **Method** | **Execution Time** | **Pros** | **Cons** |
| --- | --- | --- | --- |
| Bash Script | 27 seconds | Simple to implement, works directly in shell | Slow for large files |
| Python: Entire File | 0.44 seconds | Fast, efficient for small-to-medium files | High memory usage for large files |
| Python: Line-by-Line | 0.50 seconds | Memory-efficient | Slower due to frequent file I/O |
| Python: Split File | 0.51 seconds | Good balance between memory and performance | Slightly slower due to additional steps |

**Conclusion**

The Python methods significantly outperformed the Bash script in terms of execution time. Among the Python methods, reading the entire file into memory was the fastest, but it may not be suitable for very large files due to memory limitations. Reading one line at a time offers better memory efficiency, while splitting the file provides a balanced approach for handling larger datasets.

**Recommendation**: For systems with ample memory, Method 1 (reading the entire file into memory) is preferable. For memory-constrained systems or very large files, Method 2 or Method 3 is more appropriate.

**Screenshots**

* **Screenshot 1**: Bash script execution and output.
* **Screenshot 2**: Python methods execution and output.

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

**References**

* Stallings, W. (2018). *Operating Systems: Internals and Design Principles* (9th ed.). Pearson.
* Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.
* Python Documentation: <https://docs.python.org/>