OPERATING SYSTEMS-I

MULTI-THREADED SOLUTION FOR FINDING VAMPIRE NUMBER

STUDENT: YASIR USMANI

ROLL NO: AI22BTECH11031

1. INTRODUCTION

This program's goal is to locate vampire numbers inside a certain range by dividing the task amongst many threads. Two smaller numbers (fangs) can be multiplied and their digits rearranged to create vampire numerals. For example, since 21 * 60 = 1260, 1260 is a vampire number.

2. DESIGN OVERVIEW

2.1. LIBRARIES USED

- **stdio.h**: Standard input/output functions, used for printing messages and reading input.
- **stdlib.h**: Standard library functions, used for memory allocation and general utilities.
- **pthread.h**: POSIX threads library, used for multithreading.
- **string.h**: String manipulation functions, used for converting numbers to strings and comparing strings.
- **math.h**: Math library, used for mathematical operations like square root and power.
- **time.h**: Time-related functions, used for measuring execution time.
- **stdbool.h**: Library for boolean functions.

2.2. FUNCTIONS USED

- **compare**: Custom comparison function used by the **qsort** function to sort characters in ascending order.

- **checkVampireNumber**: Executes function calls and other operations to check if a given number is a vampire number.
- **generatePermutations**: Generates permutation of digits for a number.
- **isVampireNumber**: Compare fangs to check if a given number is a vampire number.
- **threadFunction**: Function to be executed by each thread, responsible for checking vampire numbers in a specific range.
- **main**: The main function where the program execution starts.

3. INPUT HANDLING AND EXECUTION FLOW

3.1. INPUT HANDLING

- The range of numbers (N) and the number of threads (M) are entered into the programme via the command line.
- An error message appears if the right number of arguments is not supplied.

3.2. THREAD CREATION AND RANGE ASSIGNMENT

- The main thread opens a log file (OutFile.txt) for writing.
- A struct ThreadInfo array is used to hold the details of each newly generated thread, such as its ID, start and end ranges, and the pointer to the log file.
- The numerical range is split evenly amongst the threads.
- A thread is formed and its ID, start, finish, and log file information are set for each thread.

3.3. THREAD EXECUTION

- Every thread carries out the threadFunction, searching its designated range for vampire numbers.
- For every number in the range, the checkVampireNumber function is called.
- Using the file reference given from the main thread, the thread writes the result to the log file if a vampire number is discovered.

3.3. THREAD SYNCHRONIZATION AND FINAL OUTPUT

- Using pthread_join, the main thread waits for every thread to complete.
- The total number of vampire numbers is sent to the log file by the main thread.

3.4. EXECUTION TIME MEASUREMENT

- The clock() function is used by the programme to log the amount of CPU time required during execution.
- At the conclusion of the programme, the total execution time is determined.

4. COMPLICATIONS AND CHALLENGES

4.1. THREAD COORDINATION

- Coordinating the actions of several threads becomes more difficult when multithreading is used. Within this programme:
    - Shared Variable (count): Whenever a vampire number is discovered, each thread increases the shared variable count. Inadequate coordination may result in a racing situation and inaccurate counting. In order to mitigate race problems, the main thread uses pthread_join to make sure that every thread finishes executing before the main thread outputs the final count.

4.2. FILE ACCESS

- To prevent conflicts, sharing a file reference among threads needs to be handled carefully:
    - Log File Access: A shared log file is where all threads save their results. Multiple threads writing simultaneously may result in inconsistent or corrupted data. In order to fix this, each thread uses the file reference from the main thread (threadInfo->logFile) to record its results in a sequential manner. This may cause some file pointer congestion even if it guarantees ordered writes.

4.3. PERMUTATION GENERATION

- The technique used to determine vampire numbers includes algorithm to generate permutations of digits for a given number:
    - In order to identify possible vampire numbers, the programme produces combinations of numbers. Because all potential digit combinations must be taken into account, this results in a factorial time complexity for each number.The complexity of a number grows when repeated digits are handled. The same digit may appear more than once in the permutations, therefore extra caution is required to prevent duplication.

4.4. RESOURCE MANAGEMENT

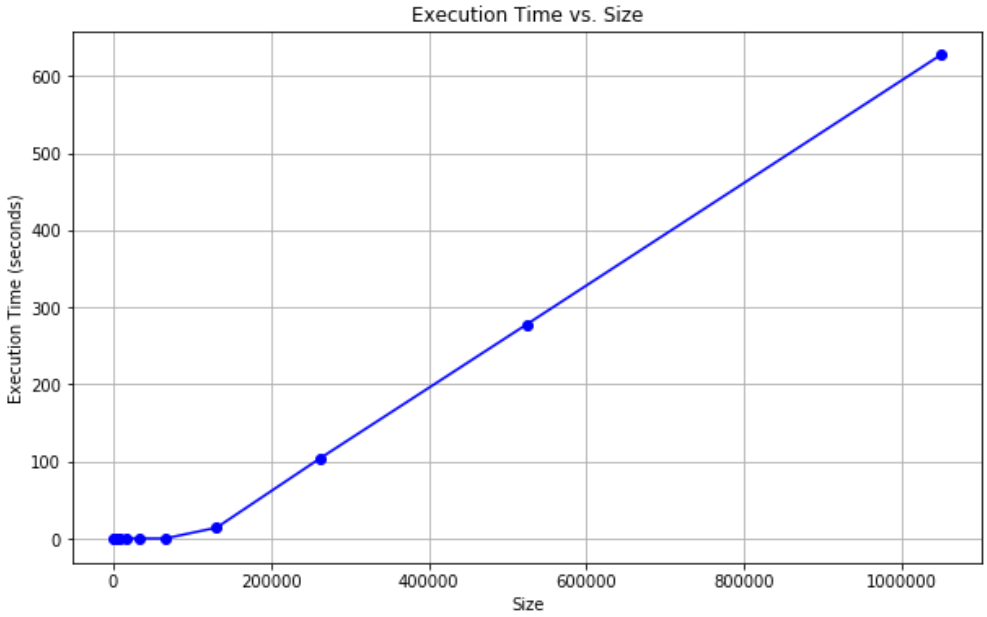
- It is crucial to allocate and deallocate resources properly:
    - **File Closure**: In order to free up system resources, the programme opens a log file for writing. It is imperative that the file be closed correctly. Although the programme makes sure the file is closed at the conclusion of execution, unanticipated mistakes might result in resource leakage.

4.5. RECURSION

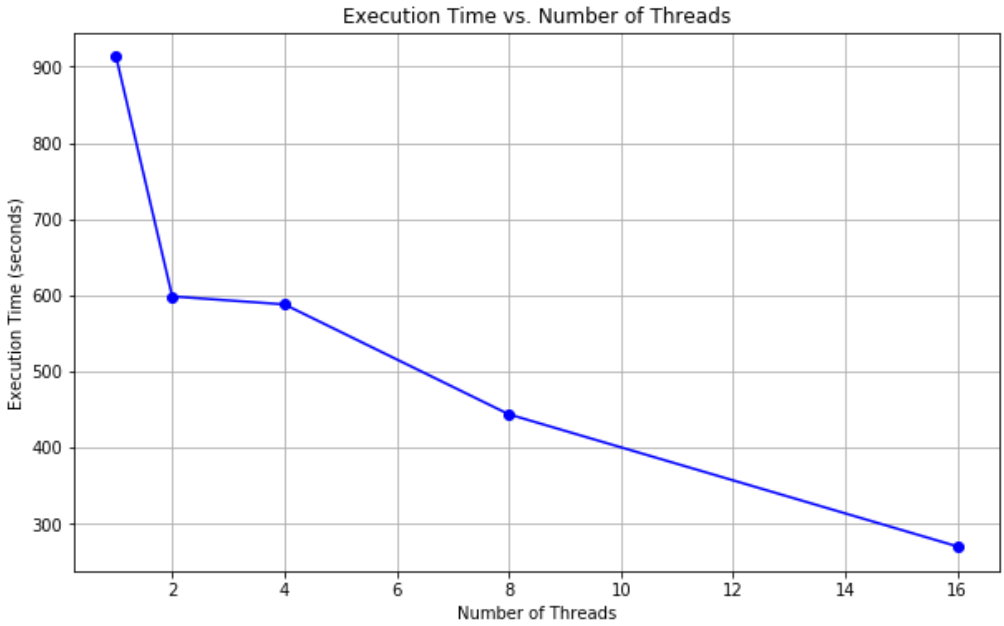
- It is crucial to apply recursion optimally:
    - Permutations are produced by recursive functions. Recursion makes the code simpler, but it can also result in a lot of function calls, which can effect memory use and even cause a stack overflow in big numbers.Performance depends on controlling recursion depth and refining the algorithm to avoid pointless recursive calls.

6. GRAPHS

6.1. TIME vs SIZE, N:



6.2. TIME vs NUMBER OF THREADS, M:



5. CONCLUSION

- The difficulties and problems that have been found draw attention to how difficult it is to create a multithreaded programme that finds vampire numbers. It will take careful consideration of shared resources, appropriate synchronization, and attention to algorithmic complexity to overcome these obstacles. Although efficient, the method of thread coordination and file access calls for a trade-off between resource management and performance. Further optimisations to improve the program's resilience and performance can entail investigating more effective algorithms or honing the thread coordination technique.