Name **: Brian Kheng**

Student ID **: A0290248B (e1324741)**

Outline and Explanation of TCP Server

- Assumptions:
  - The server binds to a specified address and port using `TcpListener::bind`.
  - Clients connect to the server and send tasks in the format `task_type:seed`.
  - The server processes these tasks concurrently, limiting the number of concurrent CpuIntensiveTask to at most 40 using a semaphore (`std_semaphore::Semaphore`).
- Implementation Details:
  - The server implements a trait `ServerTrait` with a method `start_server` to start the server.
  - The `start_server` method binds to the provided address and port, handling any errors.
  - For each incoming connection, a new thread is spawned to handle the client's request concurrently.
  - The `handle_connection` function reads lines from the client, processes the task, and sends back the result.
  - A semaphore is used to limit the number of concurrent CpuIntensiveTask, ensuring that no more than 40 tasks are processed simultaneously.

Concurrency Paradigm

- The server uses a multi-threaded concurrency model. Each incoming client connection is handled in its own thread using `thread::spawn`, allowing multiple clients to be serviced concurrently.

Level of Concurrency

- The server achieves a `Task-Level Concurrency` by spawning a new thread for each client connection. This allows multiple clients to be served simultaneously, I/O and CPU tasks from different client can also be executed concurrently which will be ensured by the operating system through context-switching.

Running Tasks in Parallel

- Yes, the server can run tasks in parallel. Each client connection is handled in its own thread, allowing multiple tasks to be processed parallelly. However, the level of parallelism may be limited by the number of CPU cores and available memory.