Server-Side:

A Socket (Treat as a File Descriptor) is made and then Bound to the given Port (User input).

A pool of Threads (The number of threads is user input) that consists of a common Buffer (Implemented as a Queue of fixed size) is made. The mutex and condition variables are initialized.

Each thread is made to run the *handleConnection* function, in which an element is Dequeued (Using FIFO Scheduling) from the Buffer of Connection File Descriptors using the Mutex as protection. If the Buffer is empty, the thread goes to sleep waits, periodically waking up and checking for the Condition Signal.
When a Connection File Descriptor is Dequequed, the name of the file to be sent to the Client is obtained from the Client. If the file exists, the file is sent along with Statistics about that request (Time at which the Request had arrived at the Server, Time at which the Request had begun to be Processed, Time at which the Request was Completed and the Thread that processed the Request).
This is repeated infinitely by each thread.

The Main Thread Listens for a Connection Request and upon receiving one, it Enqueques it into the Buffer using the Mutex as protection. A signal is raised for the Condition. The Main Thread goes back to Listening for a New Connection. This happens endlessly.

Problems Encountered:

- Since the program is endless, it has to be explicitly stopped by the user. This would cause the Port used to be Blocked for a while since the program has not explicitly freed the Port before ending. While testing the program has to be started and stopped multiple times very quickly, and using the same Port every time would lead to errors while Binding the Port since it has not been freed. Hence for testing purposes, the Port Number is taken as user input.
- Two or more threads might try to access the Buffer at the same time (Either for Enqueueing or Dequeueing) and this would lead to unwanted and unpredictable behavior. Therefore a Mutex is used to protect any Race Conditions from arising.

- If the Buffer is empty, the Child Threads have to wait for an element to be added to it before continuing. To prevent Busy-Waiting, the Condition variable is used for Signalling. The Child Threads go to sleep and periodically wake up and check whether the Condition Signal has been raised (Which will happen when an element has been Enqueued).

Client-Side:

A Socket (Treat as a File Descriptor) is made and then Bound to the given Port (User input).

The Client can perform requests in a Concurrent Manner or a Serial Manner, this depends on the user's choice.

In the Concurrent method, a number of Threads (Determined by the user) are created and each one of them connects to the Server and requests a certain file (Determined by the user). When they all finish obtaining the Data (The requested File followed by Statistics about the Request), the User is given the choice to perform this process again or exit.

In the Serial method, a number of Threads (Determined by the user) are created sequentially, the next Thread is created only when the previous Thread has finished sending its Request to the Server. A Thread connects to the Server and requests a certain file (Determined by the user) and then Signals (Using a Mutex as protection and a Condition to prevent Busy-Waiting) the Main Thread that another Thread can be created, the previous Thread then proceeds to receive Data (The requested File followed by Statistics about the Request) from the Server. When they all finish obtaining the Data, the User is given the choice to perform this process again or exit.

Problems Encountered:

- As mentioned above, for testing purposes the Server program's Port is taken as user input. Therefore this must be done on the Client-side as well.
- The global variable (Used to signify whether a Child Thread is currently still connecting and requesting from the Server in the Serial method) *callStatus*, could be accessed by both the Child Thread and the Main Thread, leading to a Race Condition. To prevent this a Mutex is used.

- In the Serial method, the Main Thread has to wait for the Child Thread to finish connecting and requesting from the Server before creating another Thread. To prevent Busy-Waiting a Condition variable is used for Signalling.