The server is started by myserver <portnumber>
For example a few terminals started as

```
./myserver 30001
./myserver 30002
./myserver 30003
./myserver 30004
```

Server is a single threaded program and can process only one concurrent client, so it initialtes the TCP listening socket with a larger backlog to allow for 20 waiting clients. The server quits if it encounter any error but the file related and commnication with client related ones. If it encounters conditions like missing file, cannot fopen or cannot seek it reponds back to the client the error condition and disconnects the client, if the error is in the very communication with the client server just disconects. Server doesnt handle any input and it must be killed to be closed. As a note , the server uses send with the flag **MSG_NOSIGNAL** to prevent it being killed by SIG_PIPE ifclient disconnects unexpectedly.

The client is started by myclient <servers text file> <number of chunks>. For example

```
./myclient server-info.text 4
```

After it reads the server ips and ports , a maximum of **MAX_SERVERS** defined as 20, the program waits for the user's input of a file to download. The input filename must be a file that exist in the erver's directory, for example the readme.txt.

Note that if the client and the server are in the same directory the client will try to overwrite the file the server is sending :) so they better be run from separate locations. So the client executable may be copied somewhere outside of the bin folder for test.

After the input of the filename the client tries to connect to one of the servers to get the file size, starting from the first till the last, if it fails to get the filesize it complains 'Cannot transfer this file'. The reasen may be both server down, or server dont have the file... if just a single server has the file and reports the file size this step is completed. The next step divides the filesize into **numchunks** chunks and spawns **numchunks** threads, each one on e different file offset and starting server index. A mutex is created to control access to the shared FILE* object.

Each thread tries to connect to a server from the list and read the data, if it could not transfer all the data it tries the next server untill all the servers are tried. When thread finishes successfully its download it marks its junk as ready, if some errors prevent the thread from completing it doesn't mark the chunk.

After all threads complete execution if some of the chunks had not downloaded the client will complain 'not all chunks are downloaded' . After that the client frees allocated resources and terminates.