Computer Networks (CS425) - Assignment 1

Directory Structure

```
├── client
│   └── ...........:: Client code
├── include
│   └── ...........:: Common header files
├── Makefile
├── obj
│   └── ...........:: Generated object files
└── server
    └── ...........:: Server code and files
```

Instructions

For compilation, execute the following from project root -

```
make
```

To start the server -

```
./server_exe <IP> <port_number>
```

NOTE: The *port_number* should be greater than 1024. The *IP* should be valid, for e.g 127.0.0.1 or your machine's IP.

To start the client -

```
./client_exe <username>:<password>@<server_ip> <port_number>
```

NOTE: If the user is not registered, a prompt will be generated for registering.
The *port_number* for the client should be the same as that of the server.
This will lead to a prompt asking for the filename. For a valid filename, the client will successfully finish execution and the requested file will be placed in the location from where the client was invoked.
The files which will be served by the server are located in the directory `server`.

Testing

For testing, execute the following commands -

```
make
./server_exe 127.0.0.1 1098
```

Invoke a new shell process, and run the following from inside it -

```
./client_exe shivansh:rai@127.0.0.1 1098  ## shivansh is a registered user
```

When prompted for a filename, enter the filename for any one of the files inside the directory `server`. For quickly checking the integrity of the transferred file, run -

```
md5sum <filename> && md5sum server/<filename>  ## should generate same values
```

Implementation Details

- The server runs in an infinite loop listening for connections.
- The maximum number of connections which can be handled **concurrently** is set to be **5** (an arbitrary value, can be updated).
- For each connection, the server `forks()` a new child process which independently handles communication with the connecting client.
- If the size of the requested file is larger than the main memory size, then it will not be possible to load the entire file at once in RAM. This case is handled by sending the file in chunks of 1024 bytes **sequentially**.
  NOTE: 1024 bytes is just an arbitrary value. To increase the speed of transfers (and reduce the number of transfers), it can be set to atmost (slightly less than) the main memory size. The disadvantage of using a small buffer size is increased I/O and socket operations.
  All the optional features are implemented.

Limitations

- **Port collisions**
  For avoiding port collisions, **port number 0** can be used which basically handles the responsibility of dynamically assigning a free port number to the operating system instead of the user. However, to keep the implementation simple, this functionality was not added.
- For looking up the pair (username, password), a linear search is done resulting in a `O(n)` running time. This can improved by loading the file `users.txt` in memory and creating a hash-set from its values. This will reduce the average search time to `O(1)`.
- The server **might** be vulnerable to buffer overflow attacks, although an attempt has been made to add safeguards wherever possible.