

**MULTISERVERS FILE DOWNLOADER**

**PROJECT REPORT**

**AMNA ABID\_\_263387**

**AMNA FAISAL\_\_263965**

**SUBMITTED TO:**

**SIR ARSALAN AHMED.**

**PROJECT SUMMARY.**

We were asked to develop a client that downloads a video file from multiple servers simultaneously.

This was implemented using simple python libraries.

**WORKING OF THE CODE:**

We made 4 virtual servers in total that can actively receive and send data to one client. For this, libraries such as *socket*, *threading* and *time* were imported. The client forms a connection with the servers at their specified port numbers and the server shows affirmation if the connection is established successfully. A few messages are exchanged between the client and the server to ensure that connection is withholding and capable of sending and receiving data. The client requests a file named as “tedtalk.mp4” from the servers that are active at the time and the servers are prompted to calculate details such as *fileSize, chunkSize*, starting point of a chunk to be sent and affirmation or negation if the servers fails to send. This is achieved by making a separate file called *“FileProcessing”* that contains different functions defined within itself for various operations. This file is imported in servers and client files. As soon as each server sends its chunk, the chunk is received by the client and recombined in the correct order of segments by using a function defined in *FileProcessing*. And thus, in this way file is sent from 4 different servers to 1 client simultaneously in correct format.

**PROBLEMS FACED:**

Problems were faced with the execution of several servers connected to one client and interacting with each other in a successful manner, and this was solved by using the concepts of multithreading. Similarly, fragmentation and conversion of a video file into a form of transportable chunks, along with functioning that was in accordance with the number of servers available, this presented another issue that was solved using programmer defined functions and the “seek” function which enabled us to divide the file into chunks and start sending from the desired point of range inside the file. The original logic of the program was to consist of an array of chunks that would be sent to each server and giving precedence servers in numeric order, the server would send the chunk at the zeroth index of the array, to the client and that chunk would be removed from that array and inserted into another array. The array of chunks would be continuously updated according to the status of servers available and upon finding the array updated or not, the server would end the transfer. Similar operations would carry out in the next server available until all the chunks have been sent and then the received chunks would be combined on the client’s side according to the array into which the chunks were being dumped and thus the file would have been received intact and in correct order.

However due to some overheads and some errors that could not be resolved till the end, the logic was discarded another commonly implemented logic was adopted that allowed similar actions to occur without the errors previously seen.