



EE-353 COMPUTER NETWORKS
PROJECT REPORT
MULTI SERVER FILE DOWNLOADER
GROUP MEMBERS:
Muhammad Asim Khaskheli (250319)
Alina Shoaib Qureshi (246312)

PROJECT INTRODUCTION:

Our task was to develop a system which can download a large sized file from **multiple servers** simultaneously.

- We had to make sure the following things:
- Each server has a separate copy of file and should operate independently
- The file should be properly fragmented, and client should download the fragments from each server based on the availability and recombine them.

MODULES USED IN THE PROJECT:

1. Os
2. Threading
3. Sys
4. Curses
5. Time
6. Socket

PROJECT IMPLEMENTATION DETAILS:

➤ MULTI THEREADING AND MULTIPROCESSING:

In order for client to connect with multiple virtual servers and employ **load balancing** among all servers we have used multi-threading and multi-processing. By using these concepts, we easily achieved the goal of connection and load balancing.

➤ FILE FRAGMENTATION AND RECOMBINATON:

Since, the requirement of project stated fair fragmentation of the file and recombination of the file on client side, we have to formulate a strategy for that too. Therefore, binary files equal to the number of servers will be created which will serve the purpose. Each server will be storing bytes in **different segments**. In addition to this, we have programmed to **delete the binary segment** after the completion of downloading process so that no extra space of user is taken. This strategy ensures the smooth sailing of the process.

➤ THE ARGUMENT PARSER:

According to the requirements of the project, we had to pass arguments along with server and client. In order to achieve this requirement, we used the “**argparse**” module which is used for the same reason.

➤ THE OUTPUT METRICS:

We had to ensure that proper output metrics are calculated and displayed. In order to accomplish this, the metrics in each process were tracked and **connected to the server using the list of ports** which served the purpose to identify the correct server in usage. In this way, the metrics were updated properly even when load balancing was applied.

PROBLEMS FACED AND THEIR SOLUTION:

1. What if any server failed? Will the bytes be consistent?


If any of the server was to fail, we had to ensure that when the work of that failed server is transferred to the other servers, the order of bytes is consistent and recombined in proper manner. In order to tackle this problem, we used the concept of sub-threading. It resolved the problem by creating segments in thread that would write back to the relevant segments in the correct order.

2. How to cope up with server failure errors on all stages?

By using the concept of exception handling, it was made sure that no errors are faced by the user and program runs smoothly.


WORKING SNAPSHTOS:

- FINDING SERVERS:

 C:\Windows\py.exe

```
Looking for available servers...  
0 server(s) available. Trying again...
```

- **SERVERS AND CLIENT CONNECTED:**

 C:\Windows\py.exe

```
Port: 12000, Status: Alive, To Shutdown Server 1 Enter E1
Port: 13000, Status: Alive, To Shutdown Server 2 Enter E2
Port: 14000, Status: Alive, To Shutdown Server 3 Enter E3
Port: 15000, Status: Alive, To Shutdown Server 4 Enter E4
Port: 16000, Status: Alive, To Shutdown Server 5 Enter E5

Kill server? Enter the key below:
```

- **OUTPUT METRICS:**

 C:\Windows\py.exe

```
Server 1: 7091/7091 bytes, download speed = 7091.0 kb/s
Server 2: 7091/7091 bytes, download speed = 7091.0 kb/s
Server 3: 7091/7091 bytes, download speed = 7091.0 kb/s
Server 4: 7091/7091 bytes, download speed = 7091.0 kb/s
Server 5: 7090/7091 bytes, download speed = 7090.0 kb/s
Total downloaded: 35454/35454 bytes
File recombination in process..
```

DOWNLOADING FINISHED:

 C:\Windows\py.exe

```
The downloading process has finished. Please press Enter to exit..
```