Multi-server File Downloader

Taimoor Ikram, Muhammad Ahsan Sajjad and Fatima Binte Tanveer
BESE-12 A

Contents

1. Abstract

This program contains functionalities of sharing a file from a server to a client machine, either on the same machine (localhost) or on another machine in the same network.

2. About the code (program)

The code features the following two components.

2.1. Server.py

This file contains the code for the server. The code is run on the machine that will send the file to the machines (clients) that will connect to it. It functions on the basis of a command line that has the following syntax.

2.1.1. How it works

The functioning is explained in the following steps.

i. The server command line (explained later) takes as argument, the port numbers of the server.

ii. The server listens on these sockets, made from the ports provided in CLI, for client requests.
iii. The server creates threads for multi-server functionality. *NOTE: Every port number entered on the client must exist first as a listening server port for it to work.*
iv. The server calculates the size of the file to send.
v. The size is then divided by the total number of available sockets. Thus, the principle of multi-server threading.
vi. Each part thus made is called a segment and is sent to the client based on the request made.
vii. Each server is requested, by using a server ID, that indicates to the server what part of the file is being requested. Accordingly, a server socket starts to send that part of the file.

2.2. Client.py

The file is run on the client machine to download a particular file from the server machine. The client machine also, like its server counterpart, features a command line that takes various arguments (explained later).

2.2.1. How it works

Given hereunder is the brief explanation of the working of the file.

i. The client takes as argument in the CLI, the same ports as the ones on the server that are currently active and have sockets running on them.
ii. After this, the client makes sequential connections with as many sockets of the server as mentioned in the command line and are running.
iii. A segment number which, in our case is the server ID, is used to request a particular part of the file from the server.
iv. All these patches of files are downloaded and saved in an array before they are written onto a file.
v. In case the download was formerly interrupted and has been initiated again, the resuming functionality has been implemented as well. A separate file that keeps a record of previously downloaded bytes and segment numbers is also kept for dealing with such situations.

3. Libraries used

The following libraries have been used for the implementation of the program.

i. Socket
ii. OS
iii. Time
iv. Threading

v. Argparse

4. Problems faced

i. During the making of this program, a number of problems were faced. One being the ability to transfer files over other devices. Using the logic of threading was good only for the server since the client failed to connect to the server outside its own premises (the server was a machine other than the client). To solve this, we tested out a sequential approach at the client side and it worked.

ii. The other problem faced was the ability to resume the downloads. The client program can save the files, keeping a track record of the segments that were previously obtained from the server along with their corresponding bytes however, the files sometimes download successfully but sometimes, they appear to become corrupted. The reason beneath couldn't be explored due to shortage of time but can be fixed in the future updates.

5. About the CLI

5.1. Server.py

NOTE: Use the command "python Server.py -h" in the directory containing this file to get the following prompt.

The server CLI has the following arguments:

usage: Server.py [-h] [-i STATUS_INTERVAL] [-n NUM_SERVERS] [-f FILE_LOCATION] -p port_list [port_list ...]

options:

-h, --help show this help message and exit

-i STATUS_INTERVAL, --status_interval STATUS_INTERVAL Time interval in seconds between server status reporting.

-n NUM_SERVERS, --num_servers NUM_SERVERS Total number of virtual servers.

-f FILE_LOCATION, --file_location FILE_LOCATION Address pointing to the file location.

-p port_list [port_list ...], --list_of_ports port_list [port_list ...] List of port numbers('n' port numbers, one for each server).

5.2. Client.py

NOTE: Use the command "python Client.py -h" in the directory containing this file to get the following prompt.

The client CLI has the following arguments:

--ver 1.0

usage: Client.py [-h] [-i METRIC_INTERVAL] [-o OUTPUT_LOCATION] [-a SERVER_IP_ADDRESS] [-p LIST_OF_PORTS [LIST_OF_PORTS ...]] [-r RESUME]

options:

-h, --help, Show this help message and exit

-i METRIC_INTERVAL, --metric_interval METRIC_INTERVAL Time interval in seconds between metric reporting.

-o OUTPUT_LOCATION, --output_location OUTPUT_LOCATION Path of output directory.

-a SERVER_IP_ADDRESS, --server_ip_address SERVER_IP_ADDRESS IP address of server.

-p LIST_OF_PORTS [LIST_OF_PORTS ...], --list_of_ports LIST_OF_PORTS [LIST_OF_PORTS ...] List of port numbers (one for each server).

-r RESUME, --resume RESUME Whether to resume the existing download in progress.

6. Learn more

Learn more about this program by using the following GitHub link. The repository can only be modified by select users.

TaimoorIkram/multiserver-file-downloader (github.com)

--ver 1.0

Appendix

A1. Getting started with this command line.

Copy and paste the following commands to your different command lines, one in one command line.

Server.py	Client.py
python -u "<directory_containing_server.py>" -n 4 -f <file_to_send> -p 4450 4451 4452 4453	python -u "<directory_containing_client.py>" -o output.mp4 -p 4450 4451 4452 4453

A2. About parser

Parser is a command line tool. It helps in making command lines so that functions can be run in the terminal without the hassle of making an easy to use command line on your own.