CS 421 Computer Networks
Programming Assignment 2 Report

Project Team:

- Osman Batur İnce 21802609 Section 1
- Burak Yiğit Uslu 21801745 Section 1

Description Of The Code

First 163 lines of the submitted code file contains the auxiliary functions defined to help with the implementation process. Code starts the execution at the 164th line, and this report also starts the discussion from there, and explains these functions as they are called in the code when necessary. Explanations for the basic functions that are also explained in the code with comments are omitted, however.

Between the lines 164 and 183, first, the number of arguments are checked, and then the necessary variables are initialized according to console input. The inputted URL of the index file is parsed to obtain a host URL, and the directory. The host is then connected to via a newly initialized socket.

Between the lines 183 and 200, a head response is sent to the host, a HTTP HEAD request is sent to the host. Depending on its response, a buffer is allocated for saving the index file.

Between the lines 200 and 228, a HTTP GET request is sent to download the index file. Here, the response message is acquired by calling the recv_all function, which was implemented for the first programming assignment originally. Fundamentally, it waits for the entire response to be received, inside a loop. It also has a timeout threshold as a safety measure. Then, the downloaded index file is parsed into the variable “response_lines” and then to the variable “file_urls”, which contains a list of URLs of the files to be downloaded. Then the socket we previously initialized is closed.

The for loop that starts at line 230 and continues until the end of the program iterates each of the file URLs in the file_urls and and downloads them using multithreading, and saves them to a file in the same directory as the program.

Between the lines 230 and 267, firstly, the variables “downloaded_file_parts” and the “thread_download_ranges” are defined. Each thread -which are called after this part- saves the parts of the file that it has downloaded to the downloaded_file_parts array. Similarly, the elements of the thread_download_ranges are also passed to the threads. Each index in the thread_download_ranges array contains the particular range that thread must download. After the initializations of these arrays, a new socket is initialized and is connected to the host of the file to be

downloaded. Then, a HTTP HEAD request is sent to the host in order to confirm that the said file exists at the said URL and is accessible. If that is not the case, by the if statement between the lines 255-257 we skip to the next iteration with the next file. After this, the socket used for sending the HEAD request is closed. Finally, the thread_download_ranges is initialized with the appropriate values, based on the length of the file (variable content_length) and connection count, by calling the function get_thread_ranges. This function calculates the ranges of the thread such that no range is more than 1 byte larger than the other for a particular file, as desired in the assignment.

Between the lines 267 and 281, first, a lock is created to be later passed into the threads, in order to prevent race conditions. Then inside the for loop between the lines 272-276, each thread is created and executes the download_file_part function, which downloads the part of the file that particular thread is supposed to download using HTTP GET requests with ranges. The way this function operates is very similar to the way "download with range" requests in the previous programming assignment works. The major difference is that once the desired data is downloaded, instead of returning the said data, the data is appended to the downloaded_file_parts_variable. The lock passed to the threads is used in order to prevent any race conditions. After execution, all threads are waited for and then terminated using join().

Finally, between the lines 281 and 297, the downloaded_file_parts are joined into a single string and are saved as a txt file.

Notes Regarding The Implementation

One important point to note about our implementation is that in the general case where the file to be downloaded has a size smaller than the thread count, the threads whose count exceed the file size (as in the 12th and 13th thread for an 11 byte file), the said threads have their ranges printed as "starting_point:starting_point-1 (0)".

An example, where a file with size 11 bytes is downloaded using 13 threads is the following:

```
1. www.cs.bilkent.edu.tr/~cs421/fall21/project1/files2/dummy1.txt (size
= 11) is downloaded
File parts: 0:0(1), 1:1(1), 2:2(1), 3:3(1), 4:4(1), 5:5(1), 6:6(1),
7:7(1), 8:8(1), 9:9(1), 10:10(1), 11:10(0), 11:10(0)
```

This is left this way after consultation with the TA, because it allows for distinguishing between cases where 1 byte is downloaded and no bytes are downloaded. (If the output was 11:11, it would be the same as the case of 10:10 for example, even though the first thread has downloaded nothing and the latter thread in this case has downloaded 1 byte.)