CS421 Programming Assignment 2 Report

Berdan Akyürek
21600904

The assignment is implemented similar to Programming Assignment 1. In the assignment, a program called ParallelFileDownloader is implemented. It is similar to the FileDownloader implemented in Programming Assignment 1 but this time it downloads a file at the same time using multiple threads.
The program is called with two command line arguments, url and connection count. Url specifies the url of the index file containing 1 or more file urls, where each line has one url. Connection count specifies the number of threads that will share a single file to download.
The implementation is done using Python3 by using sys, socket and threading libraries. Sys is used for command line arguments, socket is used to communicate with HTTP servers and threading is used to download parallelly.
The implementation is done on top of the implementation of Programming Assignment 1. The same functions are used with some modifications.
For easier management, the program is splitted into five functions and one main script. The function definitions and their explanations are given as follows:

1. `file_write(text, url)`
   This function takes a file url and its text as a string. It extracts filename from url and saves the text to extracted filename.

2. `send_request(url, typ, lower = -1, upper = -1)`
   This function sends a request to get the header or the content of the file in the url. If typ is True, it is sent as a GET and if typ is False, it is sent as a HEAD request. Also it takes optional parameters as lower and upper that define the range to be sent to the HTTP server. If they are not set, it asks for all of the content.
   After sending the request, the function waits for the server response. It keeps reading the buffer until a timeout occurs or content is fully obtained.
   After obtaining all of the content or reaching a timeout, the function returns the text obtained so far.

3. `head(url)`
   This function uses send_request function and sends a HEAD request to the url specified as a parameter. and uses the response to determine the existence and length of the content on the requested url. If the server returns an error, the function returns -1. Otherwise the function returns the “Content-Length” field of the HTTP header.

4. download(url, lower = -1, upper = -1)

   This function uses the send_request function to send a GET request to the HTTP server. It returns the content after splitting from the header, if the response contains no error. Otherwise, the function returns -1 indicating that the content could not be downloaded.

5. thread_function(url, lower, upper, thread_no, arr)

   This is the function used by threads. It takes the url, lower and upper bounds of the download region of that specific thread, number of the thread and a list to save the results of the download. It simply calls the download function with the given url and lower upper bounds until it obtains a result other than -1.

   After obtaining a correct response, thread saves its value to the given list with the index reserved for that specificthread. So that it is possible to obtain full content by merging the list, after all threads are finished.

6. Main script

   The main part of the code downloads all files by calling functions explained above.

   First, it gets and validates the command line arguments. If there is a problem with the input, it returns an error.

   Otherwise, the script prints the command line arguments to the console and it calls the download function to download the index file given as a command line argument. If the index file is not found, it returns an error.

   If the index file is found, the main script iterates over each url one by one. For each url, it calls head function to get the existence and length of that specific file. If the file is not found, it prints an error and continues to the next file. If the length is zero, it saves the file as an empty file by calling file_write function without the need of any threads.

   If the file exists and has length greater than zero, the script creates a list called parts with length same as number of threads. This list will contain parts downloaded by each thread when all threads are done.

   For each thread, script calculates lower and upper bounds according to the “Assumptions and hints” part of the assignment. Later it creates and starts each thread with the lower and upper bounds calculated. It prints the necessary logs to the console and waits for all threads to finish.

   Lastly, after every single thread is finished, the list named parts contains parts from each thread. Script simply merges all parts together as a single string. Lastly, it writes the resulting string to the required file using the file_write function explained above and moves to the next url in the index file, if it exists.