IT308 Operating Systems

Multi-Threaded Download Manager

Naitik Dodia - 201501177
Kaushal Patel - 201501219

Project Report

Introduction:

A Download Manager is basically a computer program dedicated to the task of downloading stand alone files from internet. Here, we have created a simple Download Manager with the help of threads in Python. Using multi-threading a file can be downloaded in the form of chunks simultaneously from different threads. To implement this, we have created simple command line tool which accepts the URL of the file and then downloads it. Then this command line tool is integrated with a simple GUI tool so that we can make the process user friendly.

Description:

The python libraries that we have used in this project are:

1) click - to take the input form command line “Command Line Interface Creation Kit”
2) Requests - to create a connection and read file from the server
3) Threading - To make the code multi-threaded
4) tqdm - to get the progress of the loops that are running
5) time - to calculate the time of the download
6) tkinter - to create the GUI for the downloader
7) os - to make the system call to the the command line version of the code

Serial Method:

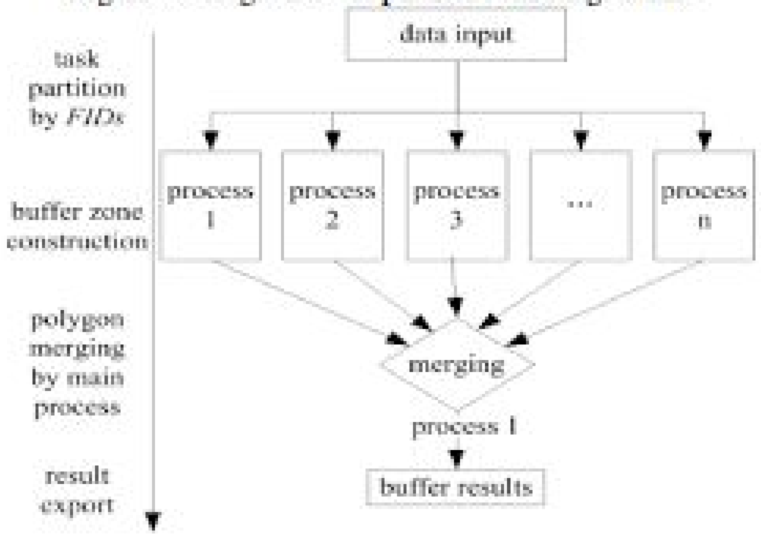
- This method is simple based on requests, tqdm and time libraries. In this method, first we define data chunk size to be 1KB = 1024 bytes.
- Then we get the url of the file which we want to download from the user.
- From this URL we generate a request response object by making a get request to the URL mentioned.
- Then we get the file size from the request response object which is stored in the headers attribute of the object.

- Then we open a file with a given name and read-write permission. Here the write permission is given with the condition that at each step the write happens by copying Byte to the file. Here, with this permission we define Byte as an atomic or indivisible object size.
- As this file is opened we copy the get the chunks of the file from the server by using the request response object that we created earlier. This is possible because request response object has a method called “iter_content()” through which we can iterate through the content of the file from the starting point to the ending point serially which can be seen as having an iterator on a list object.
- As we have used “with” statement to open the file it automatically closes the file in any unfortunate condition of exiting the process or when the download is complete. This is very much necessary because if in any situation such a condition happens:the file remains opened and the process is stopped, then hold and wait occurs and there is a higher chance of creating a DEADLOCK. So closing the file is extremely important before leaving the process.

Multi-Threaded Method:

Basic Idea behind Multi threading is Data decomposition:
The whole data is data is divided into segments which are given as a task to the threads and after execution of all threads the data is recollected from each thread into a master thread. The whole process is shown in figure below:

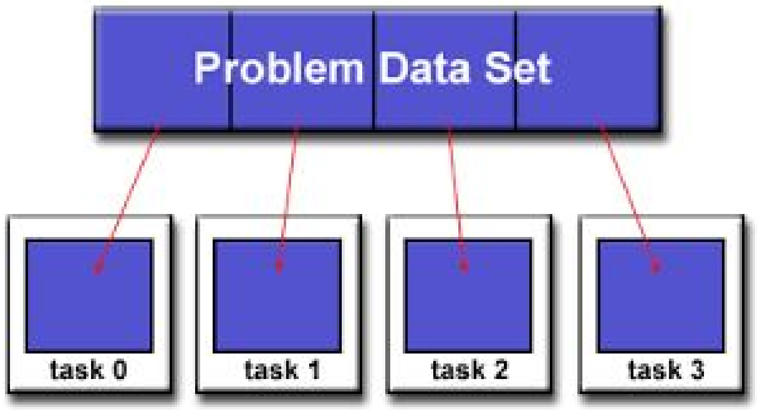
Figure 7 – Logic flow of parallel buffer algorithm.



The output of this method remains the same but the implementation and the algorithm are very different from the above mentioned serial method. In this method we taken the arguments from command line using the “click” library. These arguments include:

1) URL
2) number of threads
3) name of the file in which we have to store the downloaded file.

- In this method the part upto getting the file size remains the same as the serial method.
- After this we have to dive this file size by the number of threads that we are going to implement. This is because we have to divide the workload equally between all the threads.
- We create the file by opening the file using the same method as in the serial method. As we are not downloading the file serially the file will be written at different positions at the same time in which, the case will be such that the writing position of the current thread doesn't follow a written block.
- So, to overcome this problem we initialize the file by null-characters("\0") and the number of null character by which we have to initialize is equal to the file size in bytes. Here we have to close the file because if this file pointer remains open then it will shared by all the threads and hence it will create a disaster.
- Now as we have got the size of the file which each thread have to download we create the threads.



- The division of work is done by assigning contiguous blocks of files to the threads and this is done by assigning start index of byte and the end index of the byte.
- These two indices are passed as arguments to the handler function of each each thread. The download feature is implemented in this handler function.
- In this handler we first create a header object in which we give the range of bytes to be downloaded which are obtained from the arguments. The we create a request response object of the URL (from which we want to download the file)

with the specified handler. So this handler will contain the content which is present in the handler range.

- After getting the response object we open the file using “with” statement and give the permissions. Now we set the offset of this file pointer to the start position obtained in the argument because there will be direct mapping from the server’s file chunk position to the chunk position downloaded. Now we, write the content of the response object in the file using the specified file pointer.
- To know the file is fully downloaded we have to wait till each thread completes its task. To do this we call “join()” for each thread other than the main thread. The join() will wait for each thread to complete and it will terminate the thread.
- After every thread has completed its work of downloading we now confirm that the whole file is downloaded and we can inform the user that the file is downloaded.

GUI:

- This code consists of the GUI window that is more user-friendly than the command line interface.
- It contains the Text boxes with labels in which user can copy-paste the URL, give the name of the file and number of threads.
- This is followed by a Download button which make the system call to the CLI program with labelled arguments.

Advantages of Multi-Threaded Download manager:

- The downloading speed depends on the network and bandwidth and also the type of the chunk of the file being downloaded.
- There are some chunks of the file which cannot be downloaded with the same speed as the other normal chunks. So this creates a bottleneck for the downloading speed.
- During downloads of such files the main problem in the single threaded model is that when the threads downloads the critical chunks which are the bottleneck, it has to spend much more time compared to the other chunks. After the threads gets over with the critical chunks, it still has to download the remaining file.
- So in Multi-Threaded model as the work is divided into contiguous chunks the thread which is assigned the chunk containing the critical section will only get hinderred and other threads will take same amount of time. As soon as the critical thread completes its work other threads have already completed their

work and hence the file is fully downloaded as soon as the critical thread terminates.
- Other advantage is getting the client speed assigned by the server multiplied. Each server assigns a speed limit each request. As we are creating multiple requests we get the speed multiplied.

Disadvantages of Multi-Threaded Download manager:

- It takes some amount of time to create and join threads. This time is called the overhead of threading. So if the file is too small the overhead time overcomes the time of downloading and hence the multi-threaded model will perform poorly in comparison to the single-threaded model.
- It works well with large files.
- The speedup $\left(\frac{Serial\ execution\ time}{Parallel\ execution\ time}\right)$ gained depends on the critical chunks and position of the critical chunks. So the speedup is cannot be determined theoretically or using the number of threads used.