COL334

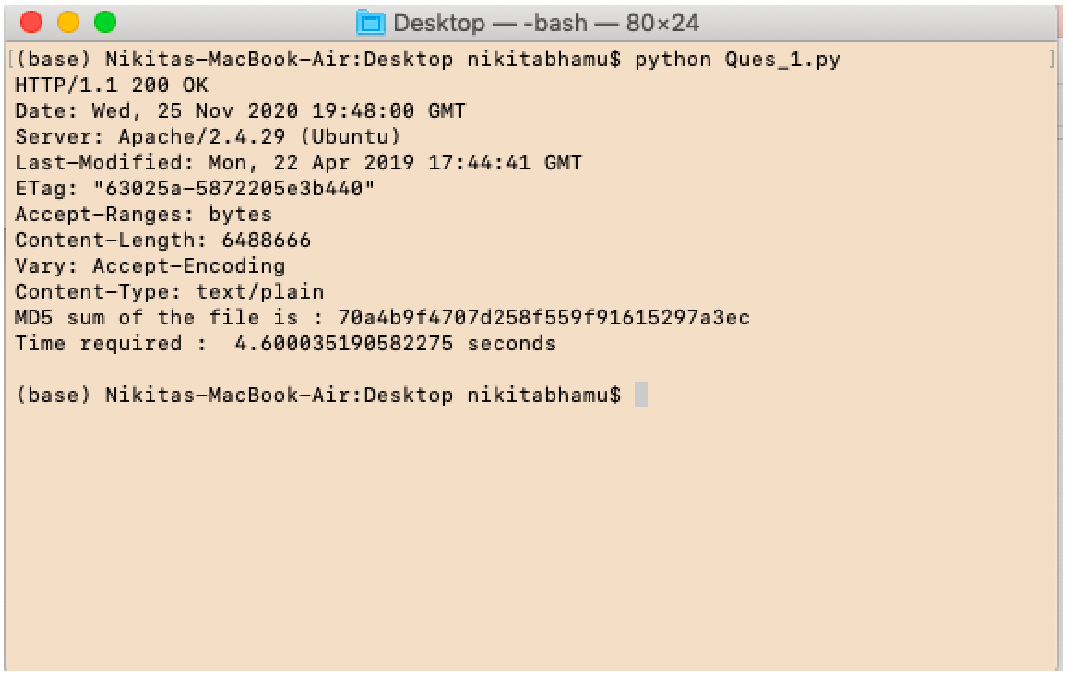
Nikita Bhamu

Assignment - 3

1 TCP socket implementation and downloading the complete file in one go

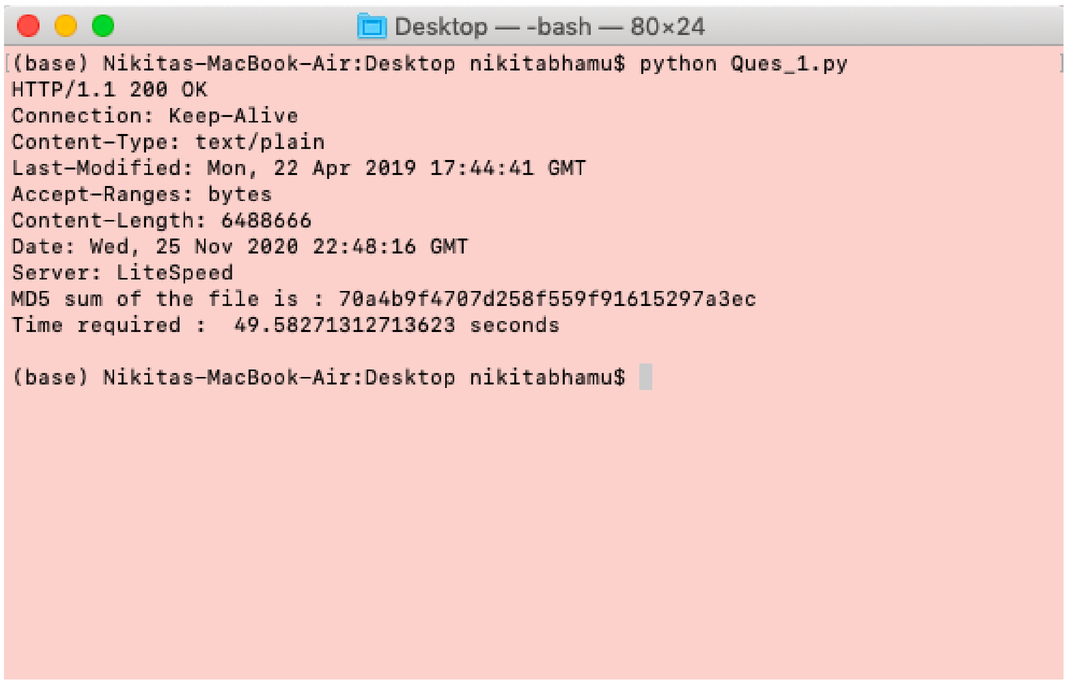
The functions of socket programming used in this are:
1. socket : It is used to create a new socket on the client side.
2. connect : It basically connects the client socket with that of the server.
3. send : It sends the request from the client to the server.
4. recv : It receives the response of the sent request from the server.

The headers of the response are removed by splitting it at ” ◦ ◦ ”.
MD5 sum of the file is calculated using the **md5 command** present in the hashlib.
For decoding the bytes received, the **decode()** function is used.
The while loop continues till the time all the bytes according to the given size doesn’t gets downloaded.
If the md5sum of the file obtained does not match with the given md5sum then all the steps are repeated. Finally when the complete file gets downloaded without any failure, then the program finishes.



```
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$ python Ques_1.py
HTTP/1.1 200 OK
Date: Wed, 25 Nov 2020 19:48:00 GMT
Server: Apache/2.4.29 (Ubuntu)
Last-Modified: Mon, 22 Apr 2019 17:44:41 GMT
ETag: "63025a-5872205e3b440"
Accept-Ranges: bytes
Content-Length: 6488666
Vary: Accept-Encoding
Content-Type: text/plain
MD5 sum of the file is : 70a4b9f4707d258f559f91615297a3ec
Time required :  4.600035190582275 seconds

(base) Nikitas-MacBook-Air:Desktop nikitabhamu$
```

Figure 1: Download file in one go from vayu.iitd.ac.in



```
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$ python Ques_1.py
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-Type: text/plain
Last-Modified: Mon, 22 Apr 2019 17:44:41 GMT
Accept-Ranges: bytes
Content-Length: 6488666
Date: Wed, 25 Nov 2020 22:48:16 GMT
Server: LiteSpeed
MD5 sum of the file is : 70a4b9f4707d258f559f91615297a3ec
Time required :  49.58271312713623 seconds

(base) Nikitas-MacBook-Air:Desktop nikitabhamu$ 
```

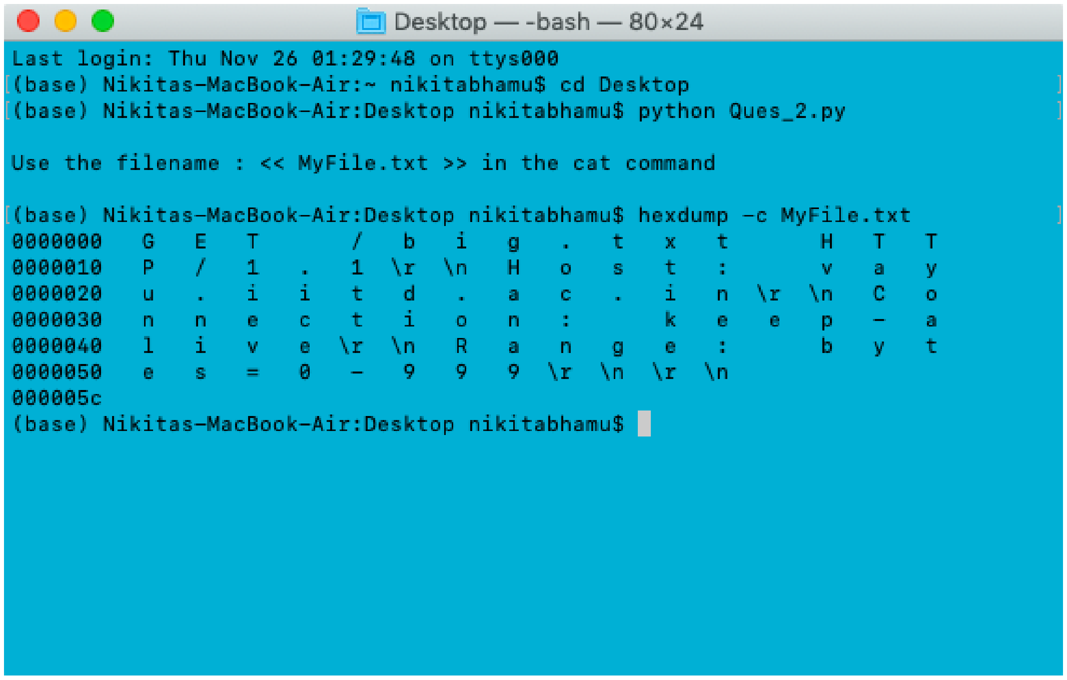
Figure 2: Download file in one go from norvig.com

The time required to download is file is much less in the situation when we download it from vayu.iitd.ac.in as compared to norvig.com, because the server of vayu is located in India, and that of norvig is located in US. So, the people surfing from India will find it faster to download the file form vayu.

2 Use of vim and downloading the file in parts

In my system I wasn't able to insert the carriage return ($\mathring{()}$) using vim even after trying all the possible combinations of CTRL,V and M. So, I made a python script in it which read a string and stores it in other file which is then passed to the cat command and then using nc (netcat) the channel to the host at the particular port is made.

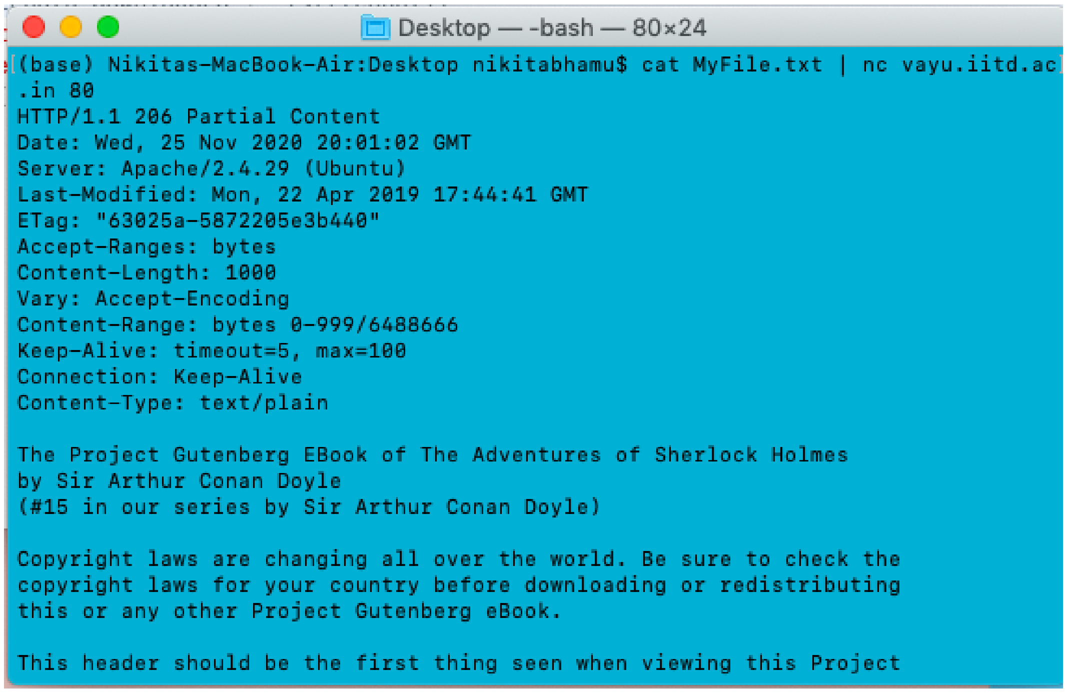
Use of the following commands:
1. cat : The cat (short for "concatenate") command. The cat command allows us to create single or multiple files, view contain of file, concatenate files and redirect output in terminal or files
2. nc : Netcat (or nc) is a command-line utility that reads and writes data across network connections, using the TCP or UDP protocols.



```
Last login: Thu Nov 26 01:29:48 on ttys000
(base) Nikitas-MacBook-Air:~ nikitabhamu$ cd Desktop
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$ python Ques_2.py

Use the filename : << MyFile.txt >> in the cat command

(base) Nikitas-MacBook-Air:Desktop nikitabhamu$ hexdump -c MyFile.txt
0000000   G   E   T       /   b   i   g   .   t   x   t       H   T   T
0000010   P   /   1   .   1  \r  \n   H   o   s   t   :       v   a   y
0000020   u   .   i   i   t   d   .   a   c   .   i   n  \r  \n   C   o
0000030   n   n   e   c   t   i   o   n   :       k   e   e   p   -   a
0000040   l   i   v   e  \r  \n   R   a   n   g   e   :       b   y   t
0000050   e   s   =   0   -   9   9   9  \r  \n  \r  \n
000005c
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$ 
```

Figure 3: Running the python script to copy the string in a text file

The output on the terminal after running this python scripts gives the name of the file which is given in the cat command. The shown above is the hexdump of text file which is found by "hexdump -c ¡filename¿"

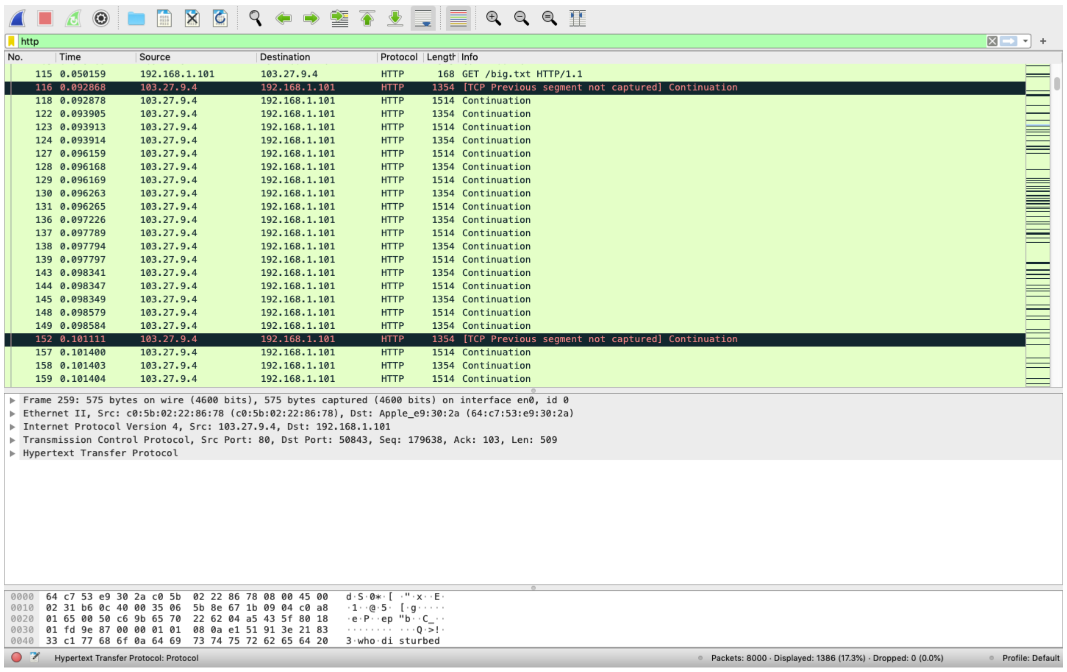


```
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$ cat MyFile.txt | nc vayu.iitd.ac
.in 80
HTTP/1.1 206 Partial Content
Date: Wed, 25 Nov 2020 20:01:02 GMT
Server: Apache/2.4.29 (Ubuntu)
Last-Modified: Mon, 22 Apr 2019 17:44:41 GMT
ETag: "63025a-5872205e3b440"
Accept-Ranges: bytes
Content-Length: 1000
Vary: Accept-Encoding
Content-Range: bytes 0-999/6488666
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/plain

The Project Gutenberg EBook of The Adventures of Sherlock Holmes
by Sir Arthur Conan Doyle
(#15 in our series by Sir Arthur Conan Doyle)

Copyright laws are changing all over the world. Be sure to check the
copyright laws for your country before downloading or redistributing
this or any other Project Gutenberg eBook.

This header should be the first thing seen when viewing this Project
```

Figure 4: Using cat and nc commands

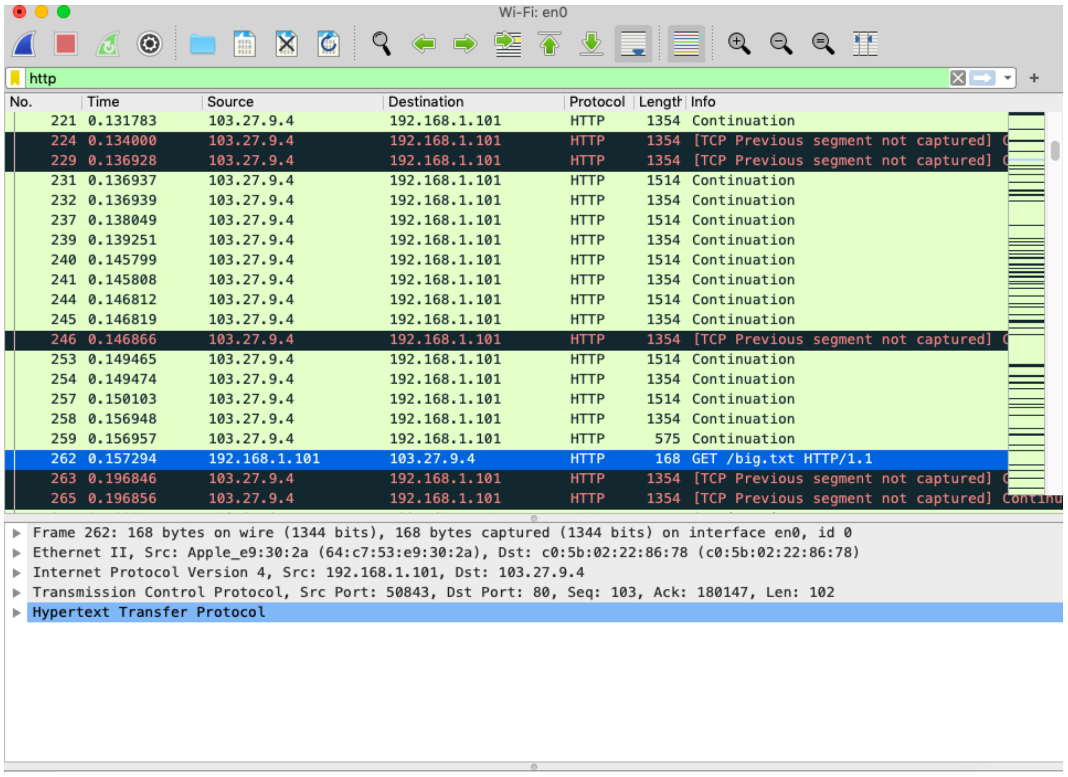
The output on the terminal gives the first 1000 bytes of the big.txt file.

Then the complete file is being downloaded by using the command of downloading the partial bytes on loops, in this 100000 bytes are downloaded in an iteration.
The wireshark trace of this download, gives around 60 responses in continuation between two consecutive get requests which are of almost 1500 bytes on an aaverage, hence it gives the size downloaded between two request = 100000.



```
No.   Time      Source          Destination     Protocol Length Info
  115 0.050159  192.168.1.101   103.27.9.4      HTTP      168  GET /big.txt HTTP/1.1
  116 0.092868  103.27.9.4      192.168.1.101   HTTP     1354  [TCP Previous segment not captured] Continuation
  118 0.092878  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  122 0.093905  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  123 0.093913  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  124 0.093914  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  127 0.096159  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  128 0.096168  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  129 0.096169  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  130 0.096263  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  131 0.096265  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  136 0.097226  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  137 0.097789  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  138 0.097794  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  139 0.097797  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  143 0.098341  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  144 0.098347  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  145 0.098349  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  148 0.098579  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  149 0.098584  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  152 0.101111  103.27.9.4      192.168.1.101   HTTP     1354  [TCP Previous segment not captured] Continuation
  157 0.101400  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  158 0.101403  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  159 0.101404  103.27.9.4      192.168.1.101   HTTP     1514  Continuation

▶ Frame 259: 575 bytes on wire (4600 bits), 575 bytes captured (4600 bits) on interface en0, id 0
▶ Ethernet II, Src: c0:5b:02:22:86:78 (c0:5b:02:22:86:78), Dst: Apple_e9:30:2a (64:c7:53:e9:30:2a)
▶ Internet Protocol Version 4, Src: 103.27.9.4, Dst: 192.168.1.101
▶ Transmission Control Protocol, Src Port: 80, Dst Port: 50843, Seq: 179638, Ack: 103, Len: 509
▶ Hypertext Transfer Protocol

0000  64 c7 53 e9 30 2a c0 5b  02 22 86 78 08 00 45 00   d·S·0*·[ ·"·x··E·
0010  02 31 b6 0c 40 00 35 06  5b 8e 67 1b 09 04 c0 a8   ·1··@·5· [·g·····
0020  01 65 00 50 c6 9b 65 70  22 62 04 a5 43 5f 80 18   ·e·P··ep "b··C_··
0030  01 fd 9e 87 00 00 01 01  08 0a e1 51 91 3e 21 83   ········ ···Q·>!·
0040  33 c1 77 68 6f 0a 64 69  73 74 75 72 62 65 64 20   3·who·di sturbed

Hypertext Transfer Protocol: Protocol    Packets: 8000 · Displayed: 1386 (17.3%) · Dropped: 0 (0.0%)    Profile: Default
```

Figure 5: Wireshark



```
Wi-Fi: en0
http

No.   Time      Source          Destination     Protocol Length Info
  221 0.131783  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  224 0.134000  103.27.9.4      192.168.1.101   HTTP     1354  [TCP Previous segment not captured] C
  229 0.136928  103.27.9.4      192.168.1.101   HTTP     1354  [TCP Previous segment not captured] C
  231 0.136937  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  232 0.136939  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  237 0.138049  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  239 0.139251  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  240 0.145799  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  241 0.145808  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  244 0.146812  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  245 0.146819  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  246 0.146866  103.27.9.4      192.168.1.101   HTTP     1354  [TCP Previous segment not captured] C
  253 0.149465  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  254 0.149474  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  257 0.150103  103.27.9.4      192.168.1.101   HTTP     1514  Continuation
  258 0.156948  103.27.9.4      192.168.1.101   HTTP     1354  Continuation
  259 0.156957  103.27.9.4      192.168.1.101   HTTP      575  Continuation
  262 0.157294  192.168.1.101   103.27.9.4      HTTP      168  GET /big.txt HTTP/1.1
  263 0.196846  103.27.9.4      192.168.1.101   HTTP     1354  [TCP Previous segment not captured] C
  265 0.196856  103.27.9.4      192.168.1.101   HTTP     1354  [TCP Previous segment not captured] Continu

▶ Frame 262: 168 bytes on wire (1344 bits), 168 bytes captured (1344 bits) on interface en0, id 0
▶ Ethernet II, Src: Apple_e9:30:2a (64:c7:53:e9:30:2a), Dst: c0:5b:02:22:86:78 (c0:5b:02:22:86:78)
▶ Internet Protocol Version 4, Src: 192.168.1.101, Dst: 103.27.9.4
▶ Transmission Control Protocol, Src Port: 50843, Dst Port: 80, Seq: 103, Ack: 180147, Len: 102
▶ Hypertext Transfer Protocol
```

Figure 6: Wireshark

3 Multithreading

As we try to download the file in chunks of size 10kB, then the pipe breaks around 104th request, this means that the keep-alive on the TCP connection is able to keep that connection alive only till the time when around 100 requests are sent, so for this I kept a count of request and when 100 requests are complete, I created a new socket and connected it with the server.

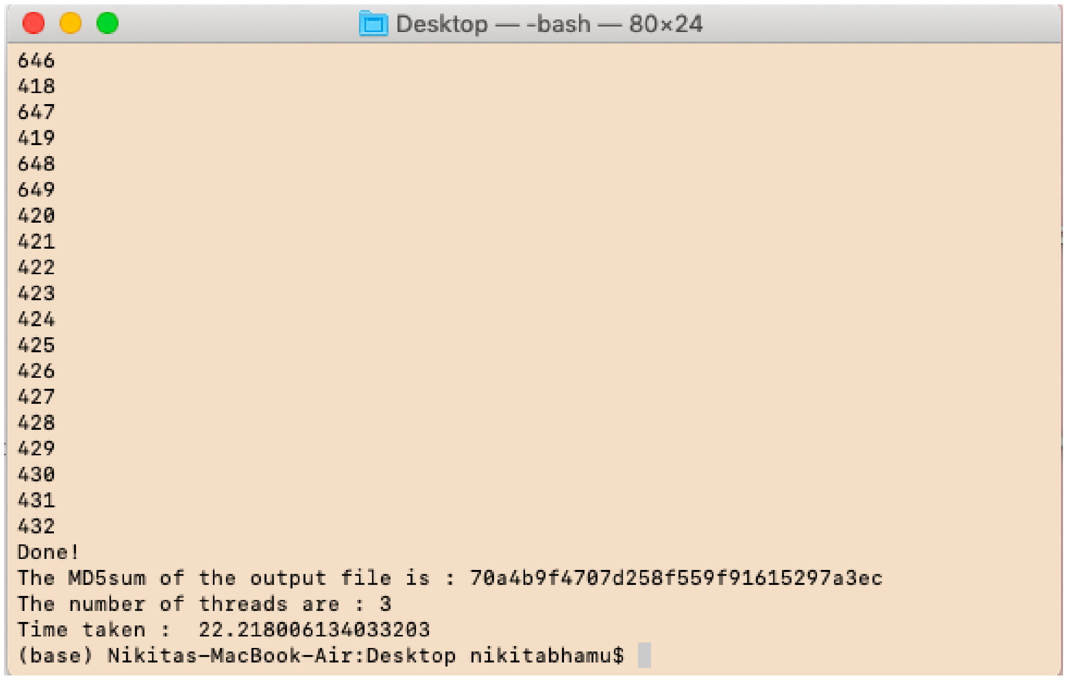
Multihtreading here means opening multiple connections with the server parallely , so that data could be downloaded parallely through those connections. These multiple connections helps us to download the file faster as we are downloading different chunks of the file parallely, hence the download speed is faster.

How this multithreading is implemented by me?
For implementing the multithreading, I implemented a function named as func which is the target of the threads. In this function the arguments are ClientSocket, the start byte, end byte, thread number, and the hostname.
The connection in a threads stays alive till the time 50 requests are sent on it,

so after that a new connection is formed to ensure smooth working.
And the data from receiver will be collected till the time the number of bytes in the range which we require does not come to us.
Along with this, the string received in each chunk is stored in an array Strings which is finly wrote in the file whose md5sum sum is matched at the end.
Along with this to plot the graph we store information about wht time a thread takes to download a chunk so that at the end the time required by each thread could be plotted.
The threads are made using the function thread-making() whihc runs a loop and passes all the arguments to the threads according to the given input and then return a list of all those threads.
Then we start all the threads of the loop and before ending this programme we wait for all the threads to join.

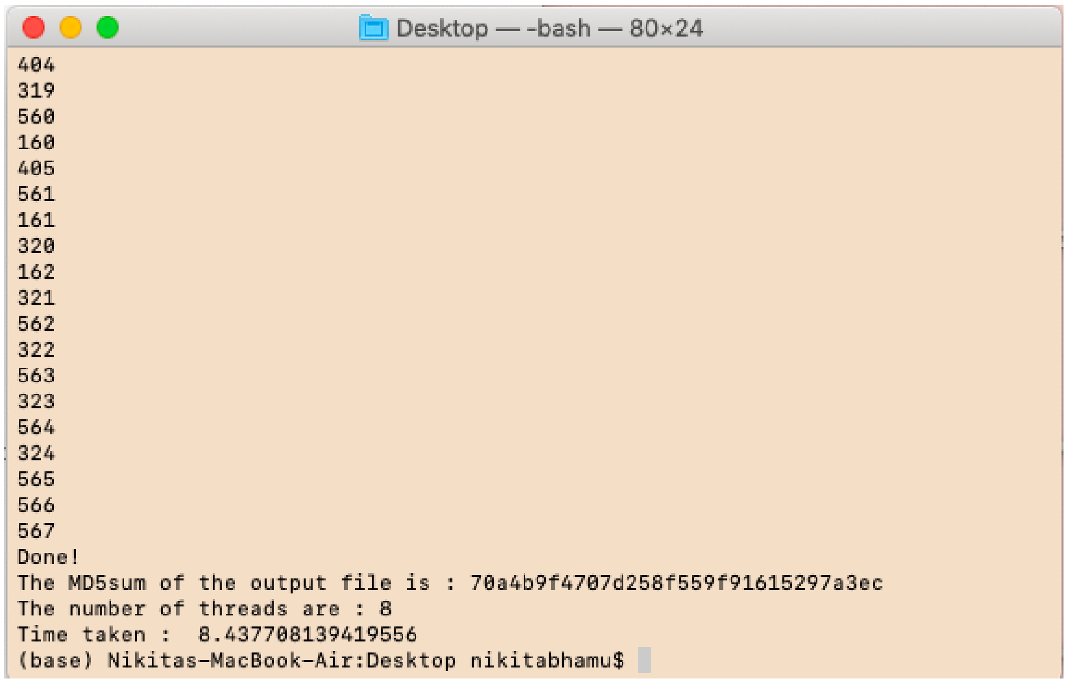
OBSERVATIONS :

1. The download times reduces with increase in number of the TCP connections but after a certain point of time (which comes after a good number of connections) it again starts incresing because the time wasted due to the handshakes in TCP connections will overpower the time saved at the time of download.
The screenshot of some of the time required are:



```
646
418
647
419
648
649
420
421
422
423
424
425
426
427
428
429
430
431
432
Done!
The MD5sum of the output file is : 70a4b9f4707d258f559f91615297a3ec
The number of threads are : 3
Time taken :  22.218006134033203
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$
```

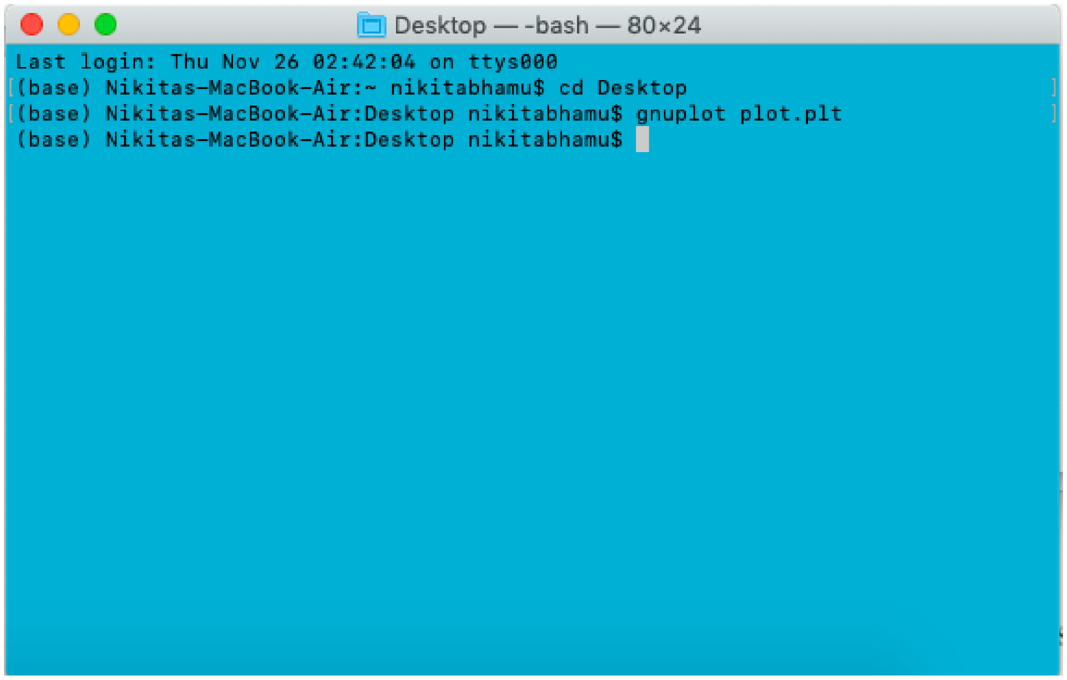
Figure 7: Time required when number of thread=3



```
404
319
560
160
405
561
161
320
162
321
562
322
563
323
564
324
565
566
567
Done!
The MD5sum of the output file is : 70a4b9f4707d258f559f91615297a3ec
The number of threads are : 8
Time taken :  8.437708139419556
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$
```

Figure 8: Time required when number of thread=8

All these observations are stored in a text file and then the graph of this information is plotted using the gnuplot.



```
Last login: Thu Nov 26 02:42:04 on ttys000
(base) Nikitas-MacBook-Air:~ nikitabhamu$ cd Desktop
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$ gnuplot plot.plt
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$ 
```

Figure 9: Gnuplot command

The following gnuplot showing the relationship of the time required to download v/s number of threads is obtained.

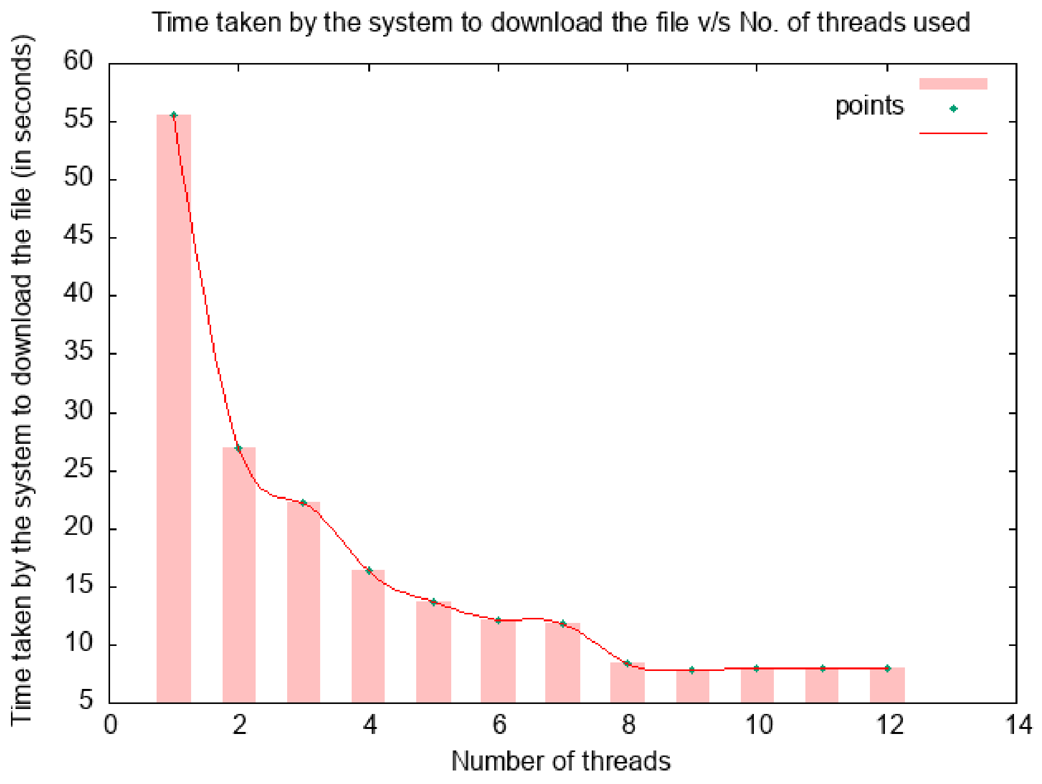


Figure 10: Plot showing the time required to download v/s number of threads

The download time in multiple connection reduces because we are parallely downloading different chucks, so time is saved due to this parellel nature of download rather than sequential.

2. The time required by different threads is on an average same unless any thread stucks in something (which is not happening in this case). The matplot library is used to plat this graph and is plotted by toring the time which all the threads require to download the different chunks. The code of this is present in the part 3 code of it.

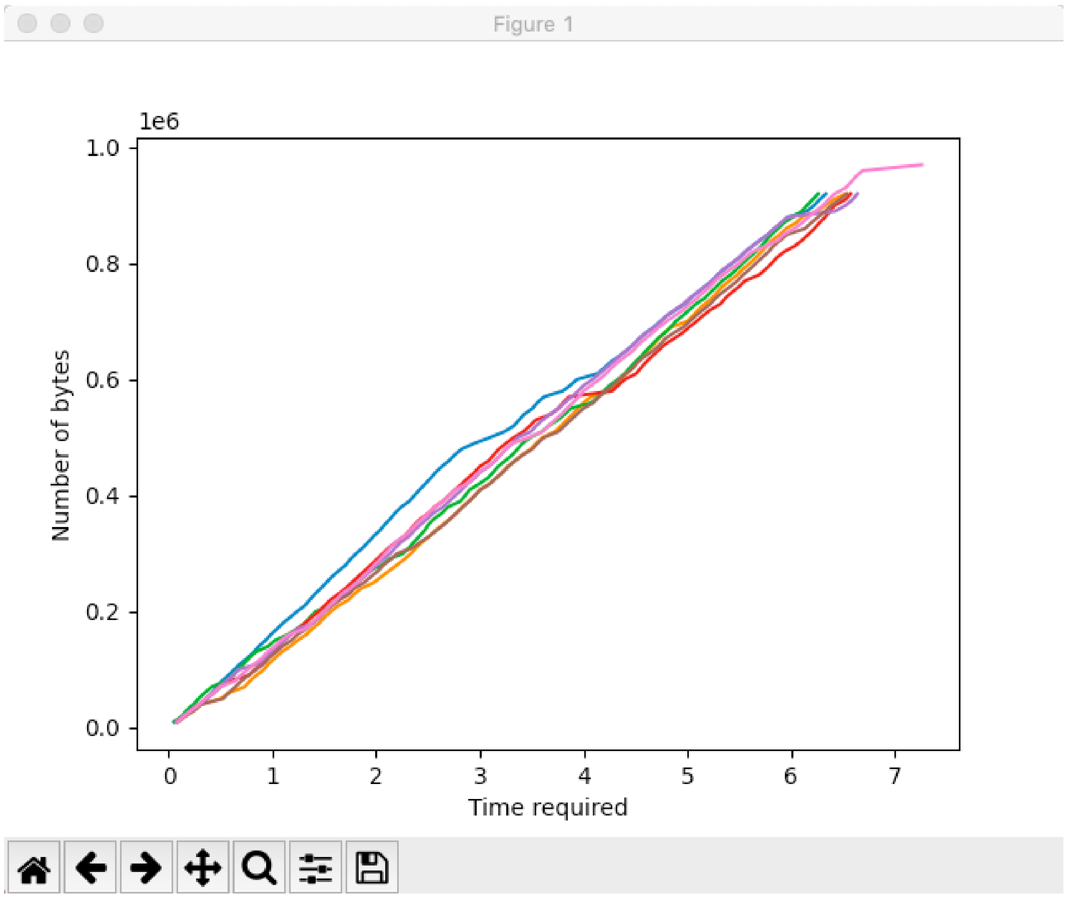


Figure 11: Download progress of 7 threads running parallelly

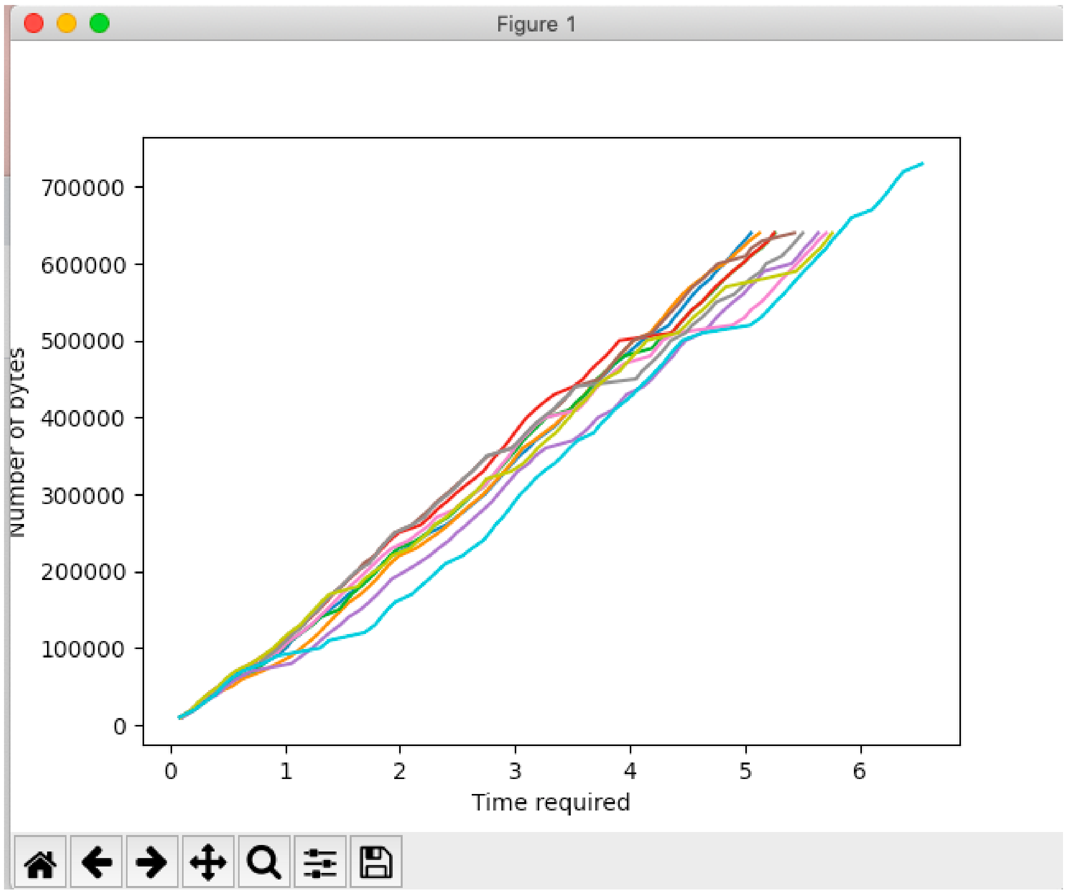
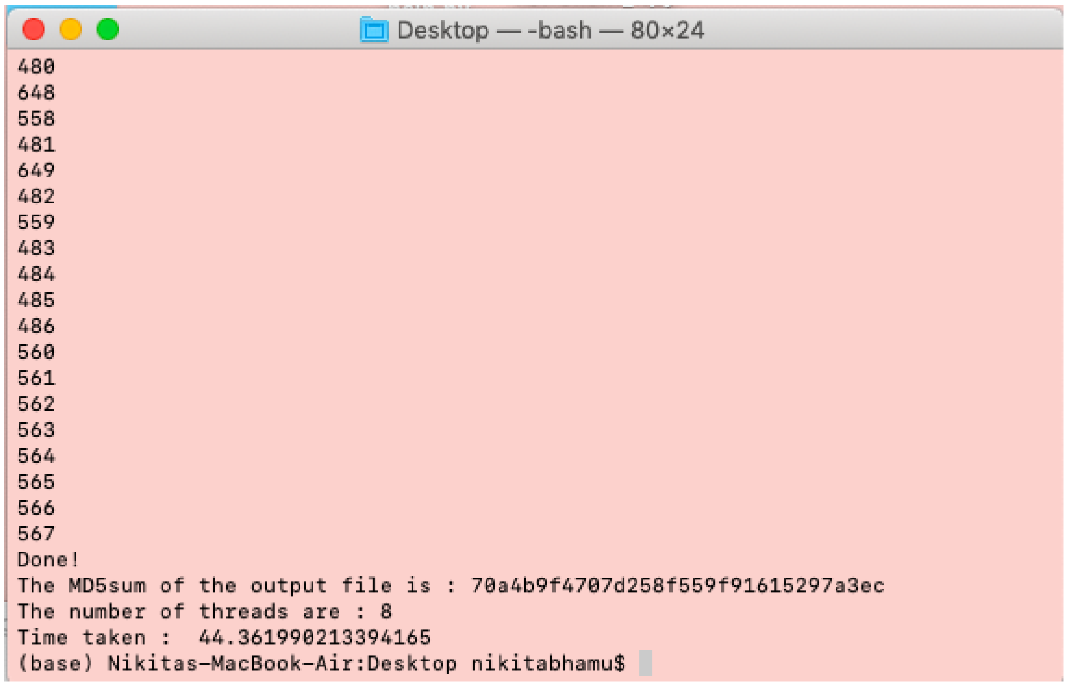


Figure 12: Download progress of 10 threads running parallelly

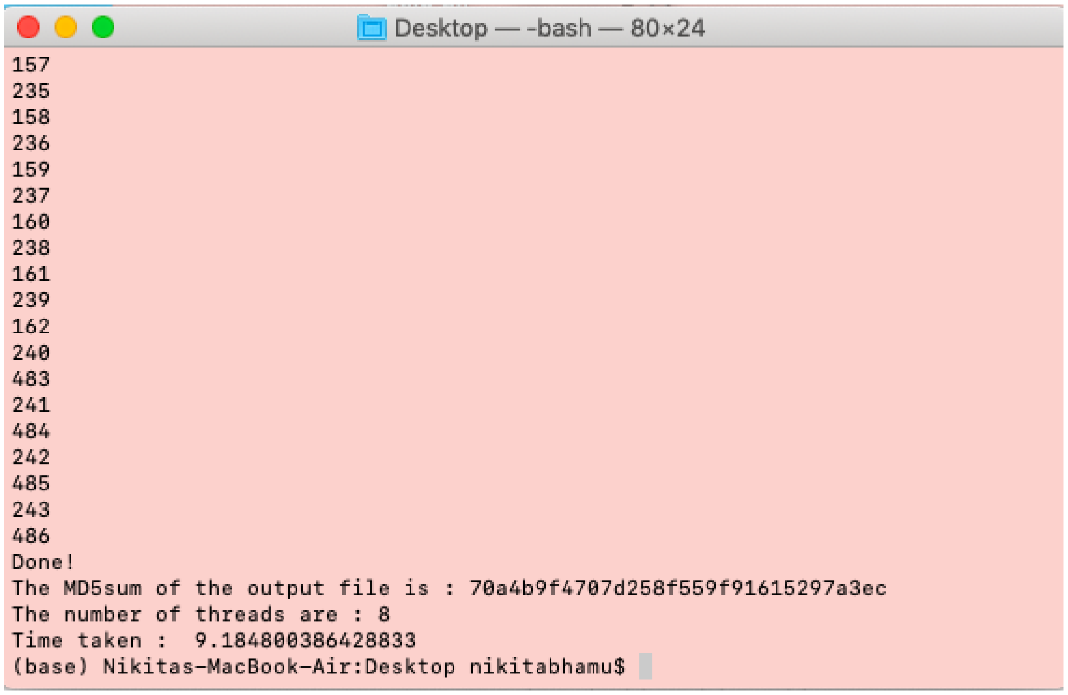
3. When we spread half of our connections to vayu.iitd.ac.in and half to norvig.com, then the download time increases relative to the time which we required when the connections were setup only on vayu.itd.ac.in and the present time is less than the time when the connection were setup only at norvig.com . So, we conclude that in a situation where number of TCP connections are fixed, then in such a situation, the splitting of those connections over other server increases only if the other servers are faster than the previous one.
The bottleneck in all such cases is decided by the slowest server among all on which the connections are spread.



Desktop — -bash — 80×24

```
480
648
558
481
649
482
559
483
484
485
486
560
561
562
563
564
565
566
567
Done!
The MD5sum of the output file is : 70a4b9f4707d258f559f91615297a3ec
The number of threads are : 8
Time taken :  44.361990213394165
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$
```

Figure 13: Time required when total TCP connections are 8, out of whic 4 are on vayu and 4 are on norvig



```
157
235
158
236
159
237
160
238
161
239
162
240
483
241
484
242
485
243
486
Done!
The MD5sum of the output file is : 70a4b9f4707d258f559f91615297a3ec
The number of threads are : 8
Time taken :  9.184800386428833
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$
```

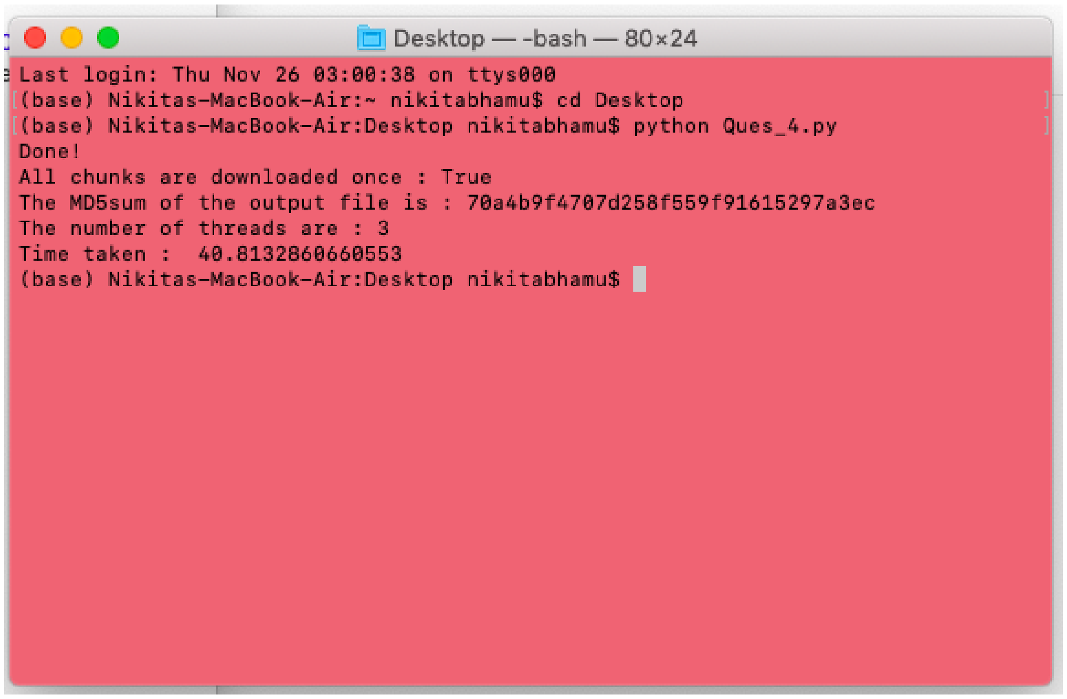
Figure 14: Time required when total TCP connections are 8, and all the 8 are on vayu

This clearly shows that the server = vayu.iitd.ac.in is faster than the server = norvig.com and hence the bottleneck in this case is norvig.com .

4 Exception handling

In this part, the connection is designed in such a way that it resumes the download of the remaining chunks even if its connection is timed-out in between due to some reason. Such an implementation saves a lot of time as after the connection resumes it only downloads the remaining chunks of the file, to see this I introduced an array which shows me the number of times a particular chunk was downloaded and I introduced the try and except function of python in the code of the 3rd question where whenever it encounters an exception, it still continues to see whether the connection is revived or not.

In this code first the timeout of the sockets is set to be 8 seconds, so that we can detect the timeout soon,
And I made a function checker which basically checks the number of times a chunk has been downloaded.
Then I stopped the connection for around 15 seconds and then resumed it again, so the file took 40 seconds to get downloaded.

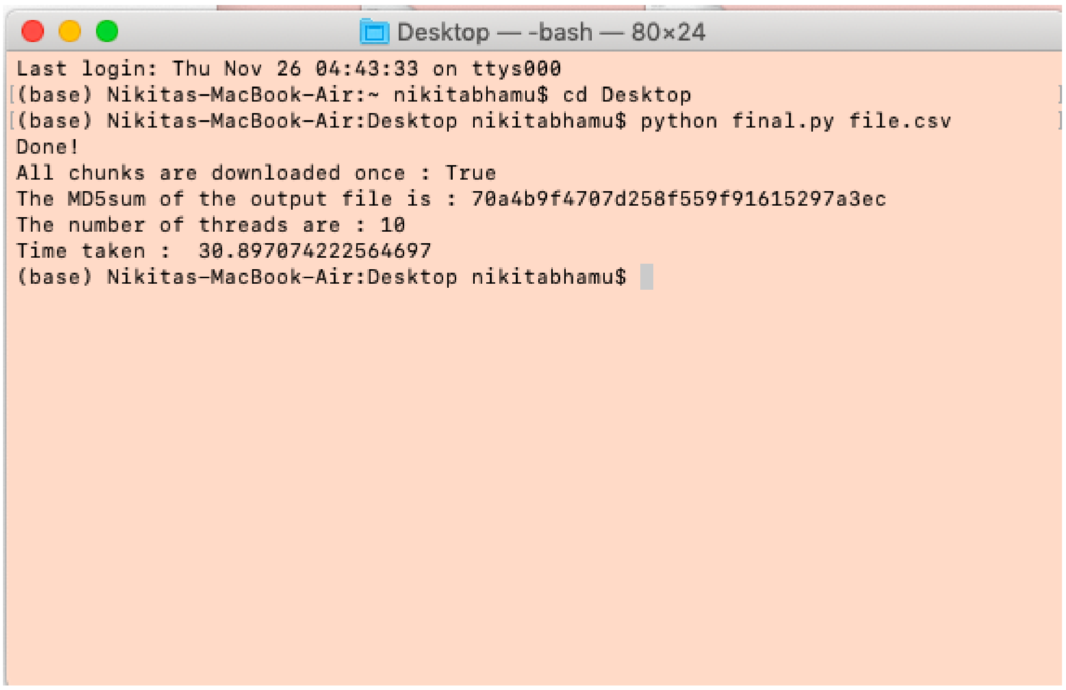


```
Last login: Thu Nov 26 03:00:38 on ttys000
(base) Nikitas-MacBook-Air:~ nikitabhamu$ cd Desktop
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$ python Ques_4.py
Done!
All chunks are downloaded once : True
The MD5sum of the output file is : 70a4b9f4707d258f559f91615297a3ec
The number of threads are : 3
Time taken :  40.8132860660553
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$
```

Figure 15: Statistics of q.4

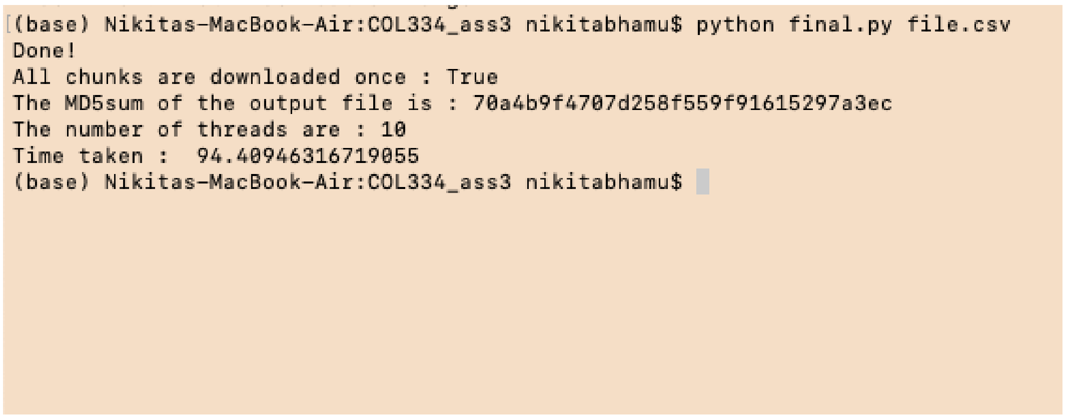
5 Final execution:

The final file named as final.py is run by giving a csv file as argument which gives the name of the two url on which the connection will be spreaded and the number of TCP connections to that particular url.
I tried to read the hostname from the csv file but due to some error I wasn't able to, so I directly gave the hostnames manually in the code of the final execution.



```
Last login: Thu Nov 26 04:43:33 on ttys000
(base) Nikitas-MacBook-Air:~ nikitabhamu$ cd Desktop
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$ python final.py file.csv
Done!
All chunks are downloaded once : True
The MD5sum of the output file is : 70a4b9f4707d258f559f91615297a3ec
The number of threads are : 10
Time taken :  30.897074222564697
(base) Nikitas-MacBook-Air:Desktop nikitabhamu$
```

Figure 16: An execution when 5-5 connections to both vayu and norvig are established and no connection distruption was there.



```
(base) Nikitas-MacBook-Air:COL334_ass3 nikitabhamu$ python final.py file.csv
Done!
All chunks are downloaded once : True
The MD5sum of the output file is : 70a4b9f4707d258f559f91615297a3ec
The number of threads are : 10
Time taken :  94.40946316719055
(base) Nikitas-MacBook-Air:COL334_ass3 nikitabhamu$
```

Figure 17: An execution when 5-5 connections to both vayu and norvig are established and connection distruption of around 12 seconds was there.