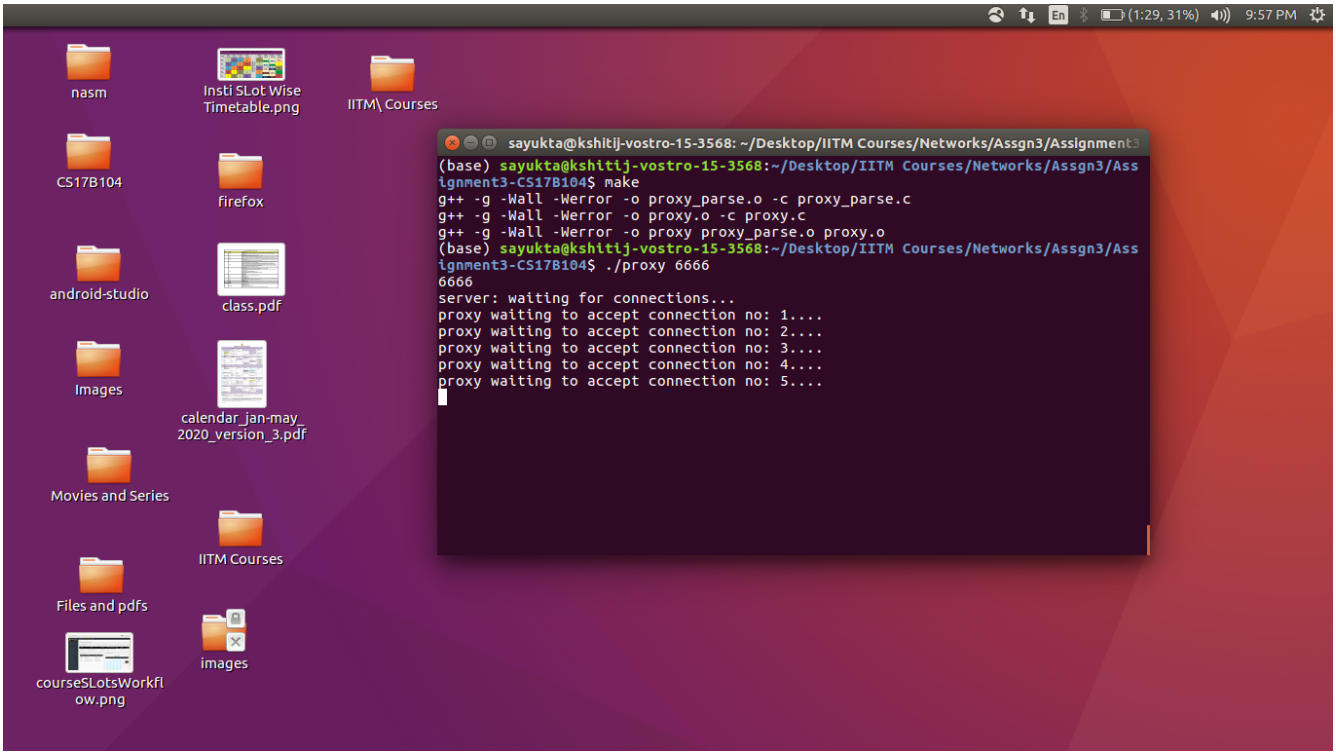
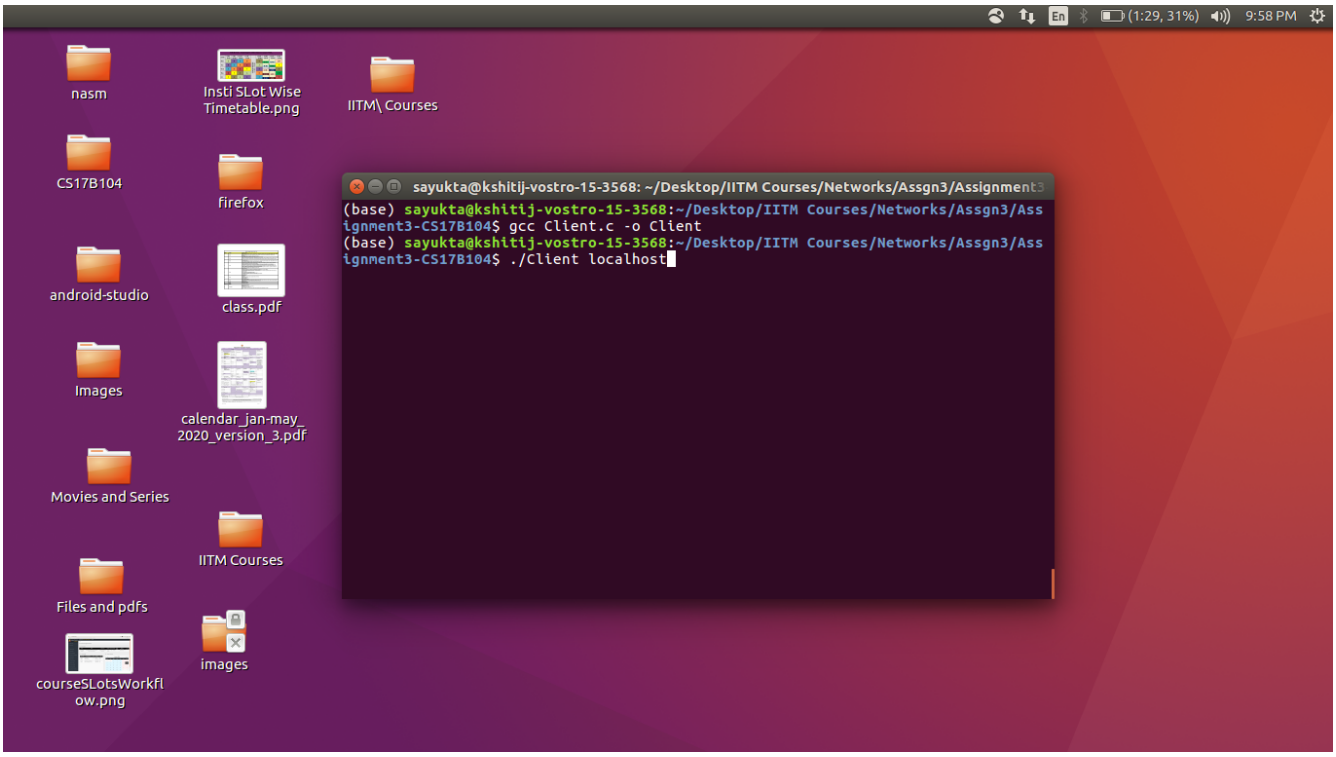
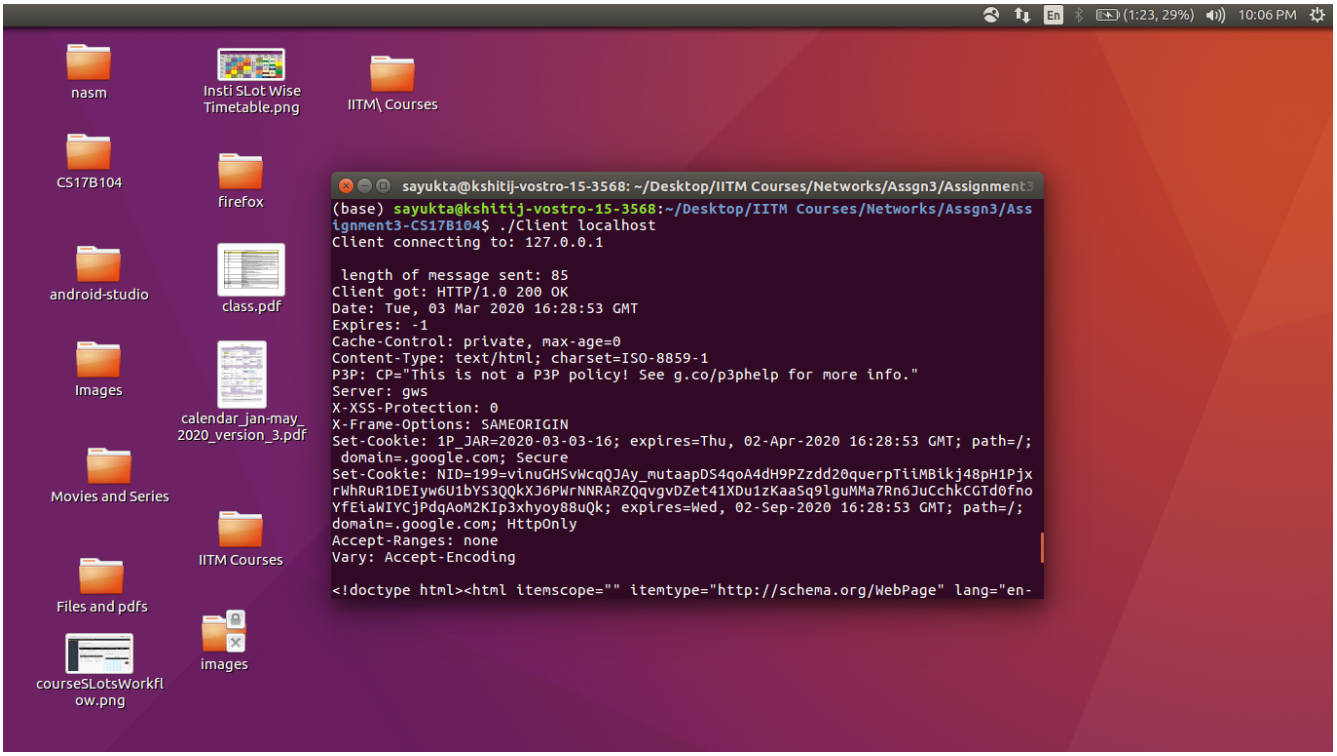
1) When you run my given proxy.c and Client.c on 2 different terminals as:

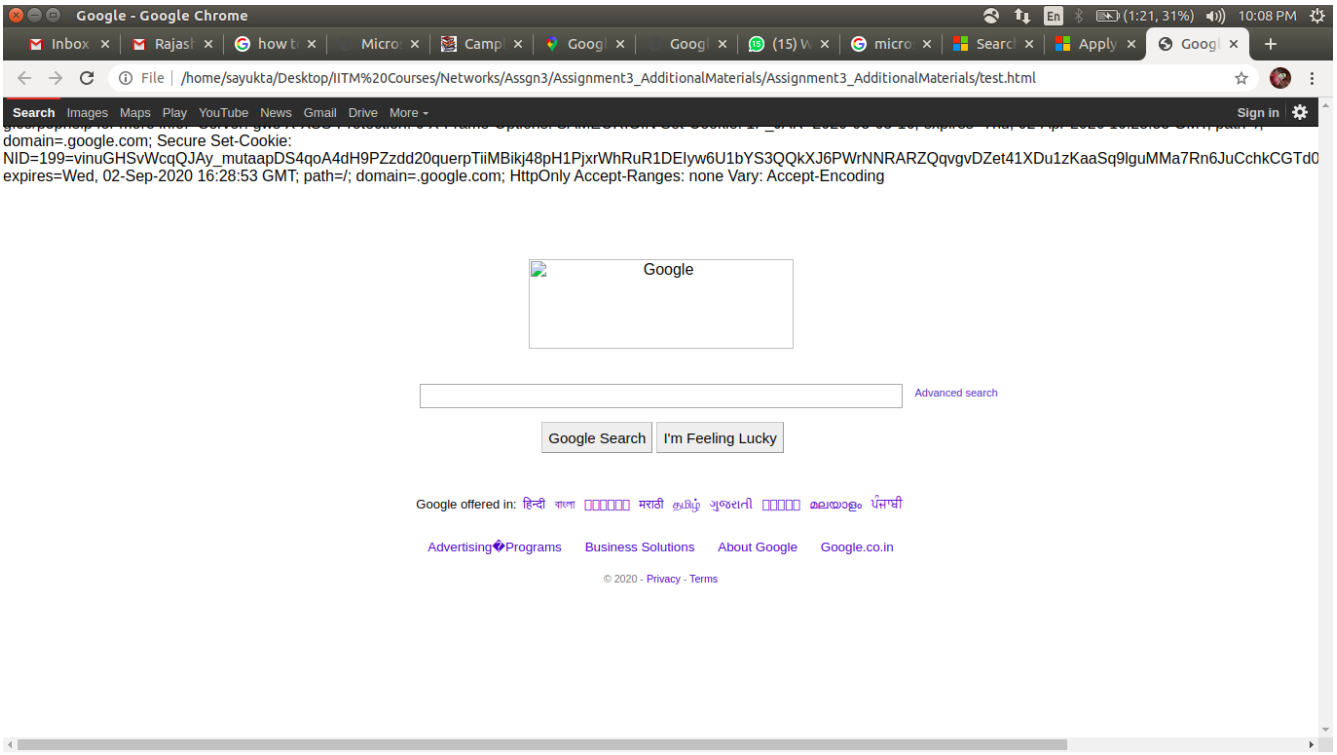




you'll get an HTML code like this on Client's terminal:



When you paste this code in a .html file, and open it using a web browser, you get:



, which is the homepage of ‘google.com’, whose link we put in Client.c on line 83. For further reference, see README.txt

This is the experiment where I run 1 client.

2) Client.c: Here, we use TCP sockets to send a GET request from Client.c to proxy.c
“hints.ai_socktype=SOCK_STREAM;” shows that as we are using SOCK_STREAM, we use TCP.
“hints.ai_family=AF_UNSPEC;” indicates that we dont care if we use Ipv4 or Ipv6.

“getaddrinfo(argv[1],PORT,&hints,&servinfo)” puts proper info in struct addrinfo* servinfo. Argv[1] has IP of machien where proxy.c is running.

“sockfd=socket(p->ai_family,p->ai_socktype,p->ai_protocol)” creates socket, based on the data we read in servinfo.
“connect(sockfd,p->ai_addr,p->ai_addrlen)” connects our 2 programs using the socket we created.

By “inet_ntop(p->ai_family, get_in_addr((struct sockaddr *)p->ai_addr),s, sizeof s);”, we can get the IP address of proxy we are connecting to.
As here its localhost, this will put in string s, the address: 127.0.0.1

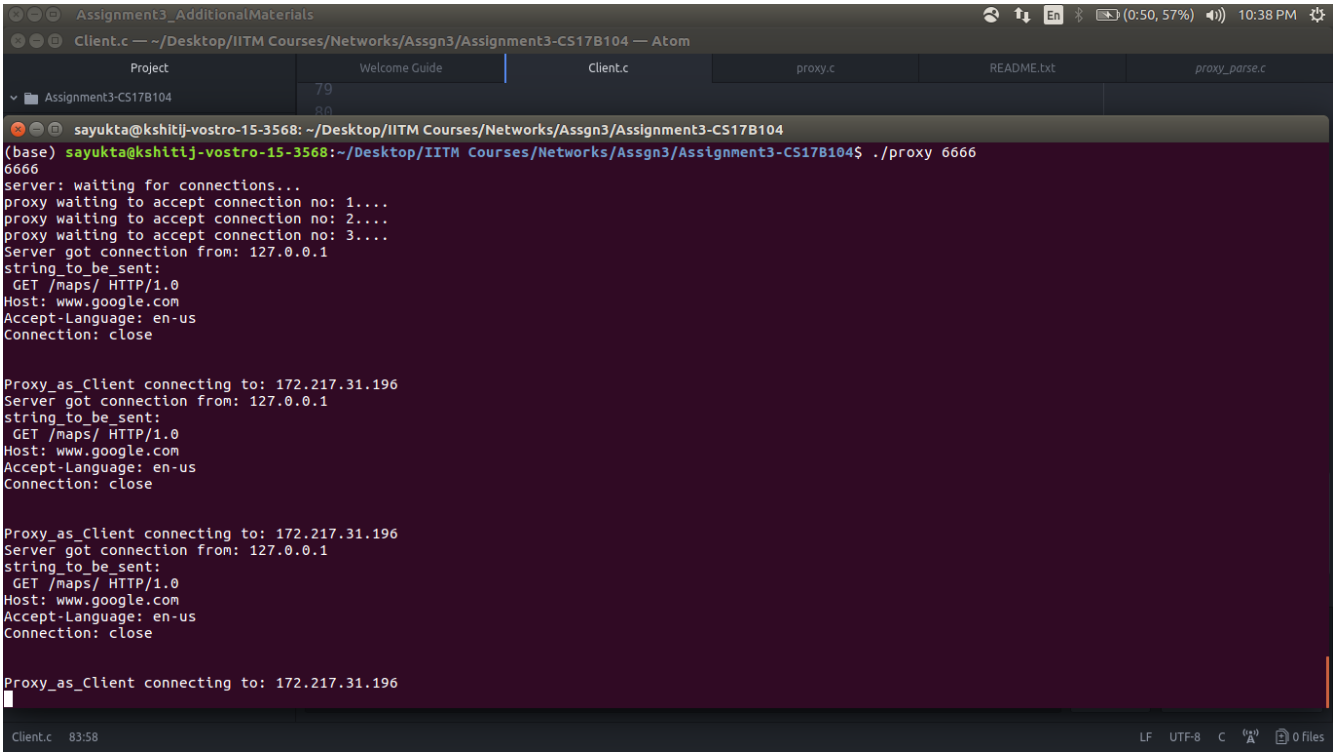
Next, we store our request string to be sent in ss, and send it via socket as:
“send(sockfd,ss,strlen(ss),0)”

Now, whatever external server data is routed via Proxy to Client has to be recieved. This is got as:
“recv(sockfd,recvv,MAXDATASIZE,0);”

3) proxy.c: Here, besides ‘socket()’, we have to call ‘setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int))’, bind(), listen(). This is because proxy.c acts as a server for Client.c, and a client for external server.
We use fork() to deal with a new Client. After this, we take the request string given by client and parse it via our libraries, to chack if its syntactically correct (else Error 400), or if its only GET request (else Error 501).
We modify this request line, using relative address in URL, and send to external servers using again, TCP sockets. We store the response got in string and forward it to Client.c using previous socket.

4) Running with 3 clients: This is experiment where I run 3 clients.
Set maximum no of connections to 3. Here, I changed URL of client to-
"GET http://www.google.com/maps/ HTTP/1.0\r\nAccept-Language: en-us\r\nConnection: Continue\r\n\r\n\0".
Run proxy first, and then, in a different terminal, run ‘./Client’ 3 times. You will find that when you type ‘./Client’ for the 4th time, proxy will give no response, as we had set upper limit of no. Of requests to proxy as 3.

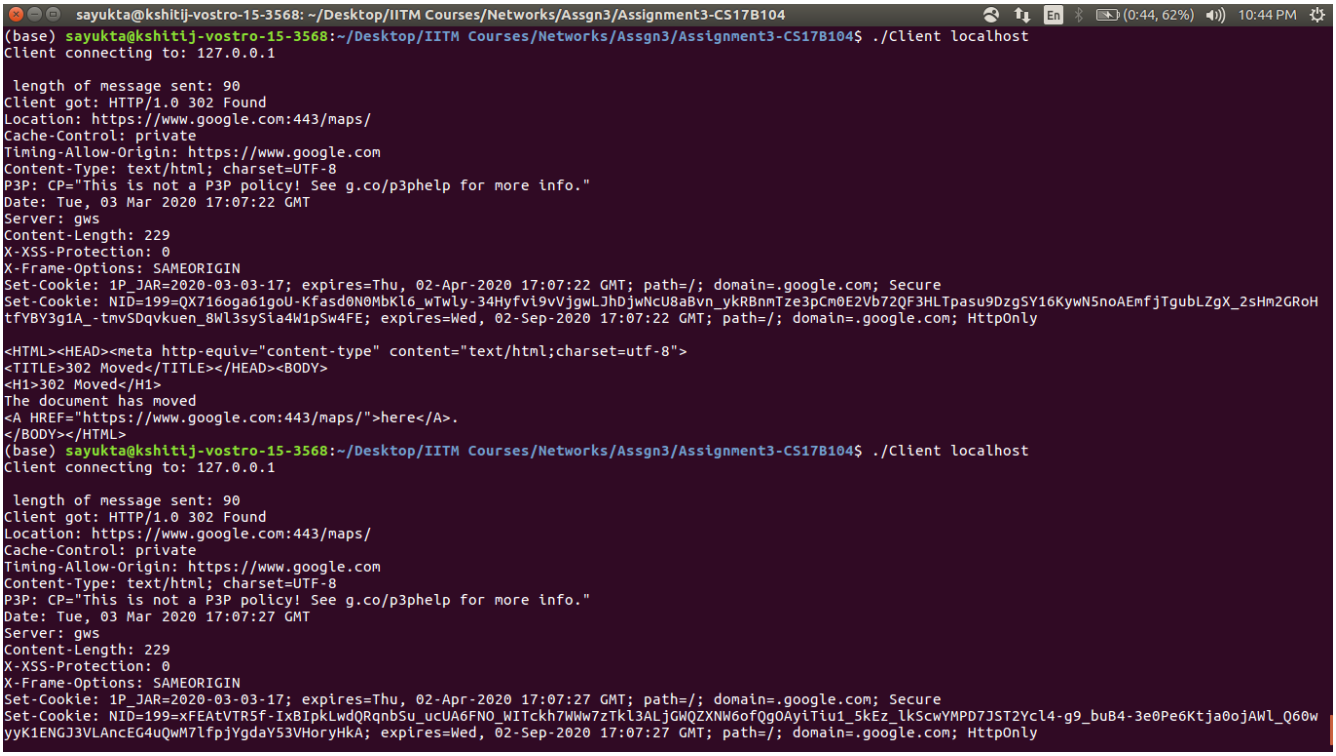
Proxy’s terminal looks like:

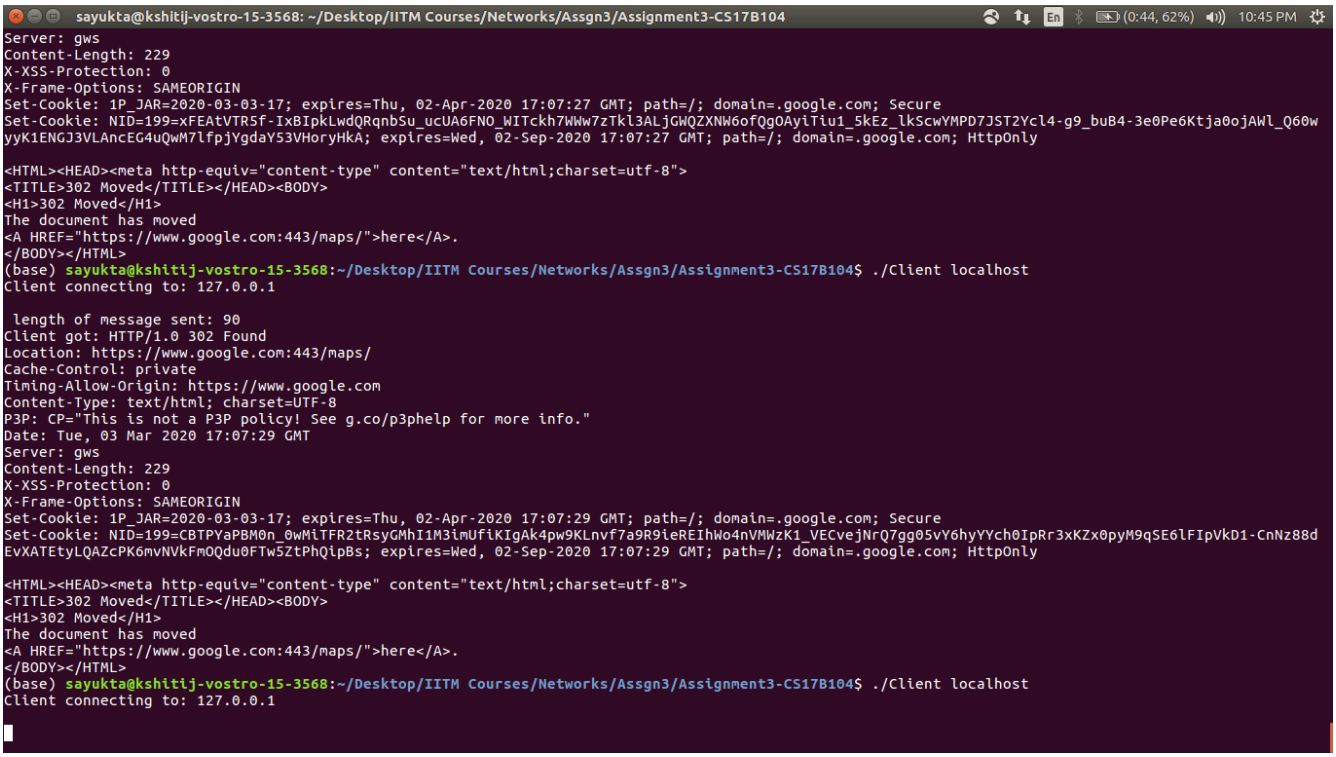


Here, Client is 127.0.0.1, so it says got connection from that.
The 172.217.31.196 is the IP address of Google server running Google maps.
Note that its run 3 times, and will wait, even if we run Client 4th time.

string_to_be_sent is the string to be sent to external server.

Client’s terminal will look as:





This represents client running 3 times. Each time, html page of google maps got by Client.
At 4th time, no response comes from Client.

4) What I learnt: I learnt the way we can make sockets in TCP, and the basic

structures involved, as I coded this in C. I learnt the various functions such as socket(), bind(), listen(), send(), recv() ,etc used. This is not a very abstracted view as compared to Java, Python etc.

The reason we use a proxy is that we put the earlier searches of user in cache, which can be got faster if he reaccesses them.
Proxy is server for client, and client for external server. Thats why we create 2 sockets in proxy, one to Client.c to get the request string, and one to external server.

Here, a point to note was that when html text data comes from external server, it doesnt come as a single string but as chunks. So, I had to use a while loop to collect it in a single string.

Also, not all programs had ‘HTTP/1.0 200 OK’ response. Some, such as Google maps showed: ‘HTTP/1.0 302 Found’, and some, such as reebok.com, even showed: ‘HTTP/1.0 301 Moved Permanently’
I suspect these maybe due to https being used, and not http.

I learnt difference in Ipv4 and Ipv6; in UDP sockets and TCP sockets and how to use SOCK_DGRAM for former and SOCK_STREAM for latter. TCP packets are safer than UDP, but UDP are faster, so are used in games, video, etc.

I think I have also found a way besides ping, to crash a server! Just, put maximum_no_of_connections as some large number, and from a client on your laptop, call proxy many times.