ECEN 602 HW 4
Programming Assignment
(Implementation of HTTP 1.0 Proxy Server and Client)

Team Number 8

1. Garuda Suma Pranavi (UIN 926009146)
2. Dedeepya Venigalla (UIN 726006161)

We have both equally contributed in developing the command line client and the proxy server.

Design:

1. The client sends a HTTP request with explicit URL.
2. When the proxy server receives a client request, it first checks its cached data to try and serve the request.
3. If there is no match in the entries, the request is then sent to the web server and the response is then relayed by the cache to the client. The proxy also stores it for later use.
4. The proxy server maintains 10 entries in the cache which is updated according to the time field.

   Note: This code has been tested and developed in the cloud9 environment. The header fields were checked through the use of Firefox Quantum: Developer edition, header fields.

   Implementation: We have checked for the following URL's:
   - www.google.com
   - www.cricbuzz.com
   - http://www.classicperform.com
   - www.columbia.edu
   - www.tamu.edu
   - www.berkeley.edu
   - www.cornell.edu
   - www.vit.ac.in
   - www.ox.ac.uk
   - www.northwestern.edu
   - www.purdue.edu
   - www.uchicago.edu

How to implement the code:

1. Please attach the uploaded makefile to compile the code:

```
all: proxyserver.cpp client.cpp
g++ -o proxy proxyserver.cpp
g++ -o client1 client.cpp
```

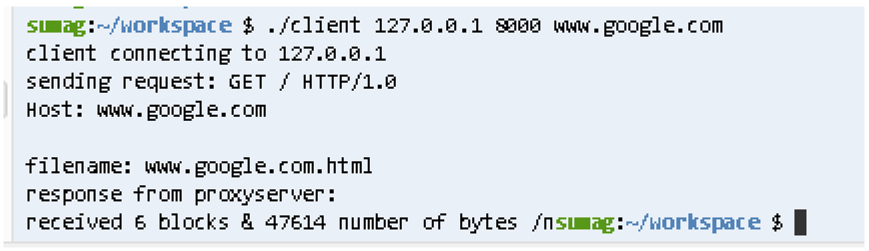
g++ -o client2 client.cpp
g++ -o client3 client.cpp
clean: server.cpp client.cpp
rm server
rm client1
rm client2
rm client3

2. How to run the code:

Proxy Server: /proxy <ip to bind> <port to bind>
Client: /client <proxy address> <proxy port> <URL to retrieve>

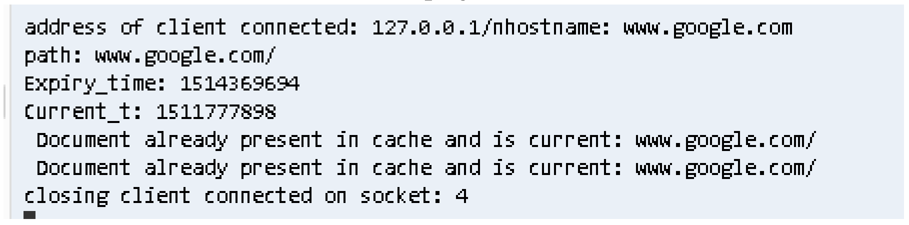
Test Cases

1. A cache hit returns the saved data to the requester



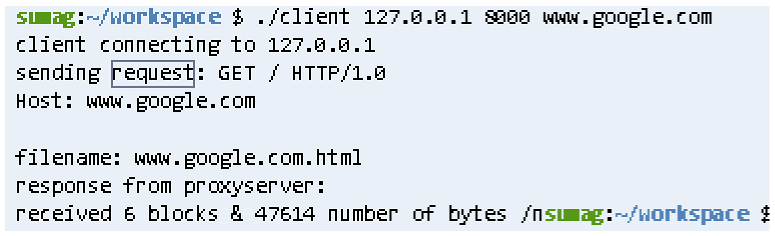
```
sumag:~/workspace $ ./client 127.0.0.1 8000 www.google.com
client connecting to 127.0.0.1
sending request: GET / HTTP/1.0
Host: www.google.com

filename: www.google.com.html
response from proxyserver:
received 6 blocks & 47614 number of bytes /nsumag:~/workspace $
```

We can see that the current_t<expiry_t i.e. the document is not stale and hence can be retrieved

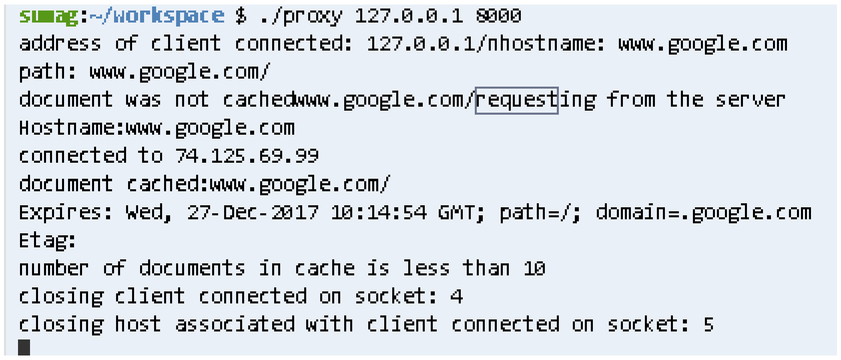


```
address of client connected: 127.0.0.1/nhostname: www.google.com
path: www.google.com/
Expiry_time: 1514369694
Current_t: 1511777898
 Document already present in cache and is current: www.google.com/
 Document already present in cache and is current: www.google.com/
closing client connected on socket: 4
```

2. A document which is not found in the cache is requested from server, proxied and then sent to the client

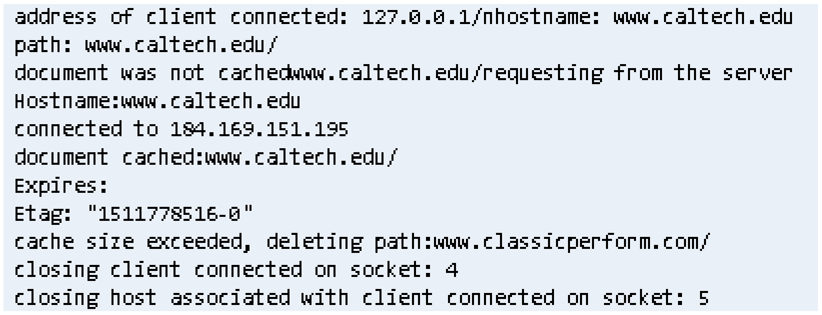


```
sumag:~/workspace $ ./client 127.0.0.1 8000 www.google.com
client connecting to 127.0.0.1
sending request: GET / HTTP/1.0
Host: www.google.com

filename: www.google.com.html
response from proxyserver:
received 6 blocks & 47614 number of bytes /nsumag:~/workspace $
```



```
sumag:~/workspace $ ./proxy 127.0.0.1 8000
address of client connected: 127.0.0.1/nhostname: www.google.com
path: www.google.com/
document was not cachedwww.google.com/requesting from the server
Hostname:www.google.com
connected to 74.125.69.99
document cached:www.google.com/
Expires: Wed, 27-Dec-2017 10:14:54 GMT; path=/; domain=.google.com
Etag:
number of documents in cache is less than 10
closing client connected on socket: 4
closing host associated with client connected on socket: 5
```

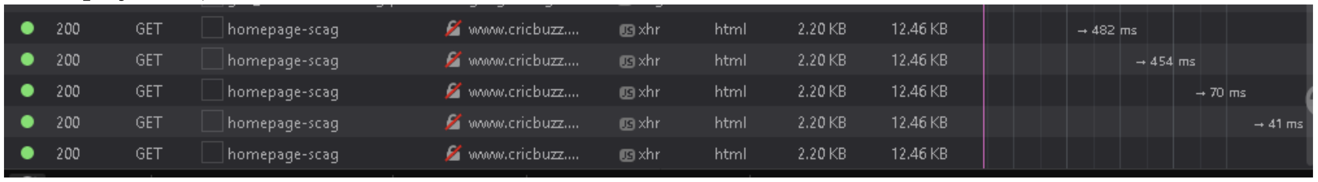
3. A cache that has 10 items, proxies the new request, removes the LRU file, saves the data received from server in the cache and relays it to the client



```
address of client connected: 127.0.0.1/nhostname: www.caltech.edu
path: www.caltech.edu/
document was not cachedwww.caltech.edu/requesting from the server
Hostname:www.caltech.edu
connected to 184.169.151.195
document cached:www.caltech.edu/
Expires:
Etag: "1511778516-0"
cache size exceeded, deleting path:www.classicperform.com/
closing client connected on socket: 4
closing host associated with client connected on socket: 5
```

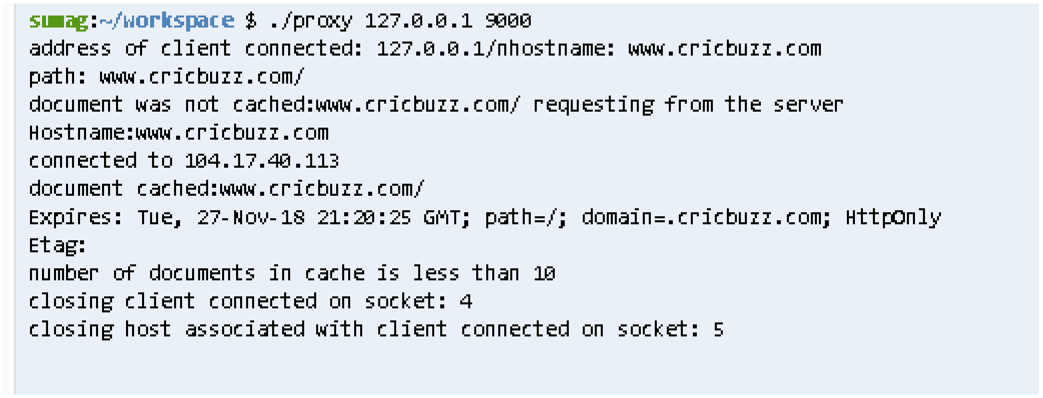
4. A stale Expires: header in the cache is accessed, the cahce entry is replaced with a fresh copy, and the fresh data is delivered to the requester.

   Network headers : (Snapshot showing cricbuzz.com page retrieval every few ms due to lesser expiry-time)



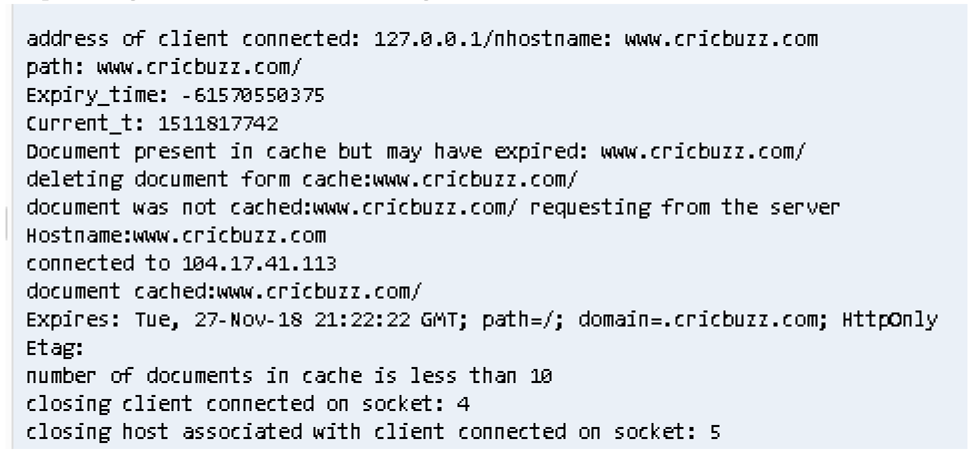
●	200	GET	homepage-scag	www.cricbuzz....	xhr	html	2.20 KB	12.46 KB	→ 482 ms
●	200	GET	homepage-scag	www.cricbuzz....	xhr	html	2.20 KB	12.46 KB	→ 454 ms
●	200	GET	homepage-scag	www.cricbuzz....	xhr	html	2.20 KB	12.46 KB	→ 70 ms
●	200	GET	homepage-scag	www.cricbuzz....	xhr	html	2.20 KB	12.46 KB	→ 41 ms
●	200	GET	homepage-scag	www.cricbuzz....	xhr	html	2.20 KB	12.46 KB	

Getting the document for the first time and storing in cache:



```
sumag:~/workspace $ ./proxy 127.0.0.1 9000
address of client connected: 127.0.0.1/nhostname: www.cricbuzz.com
path: www.cricbuzz.com/
document was not cached:www.cricbuzz.com/ requesting from the server
Hostname:www.cricbuzz.com
connected to 104.17.40.113
document cached:www.cricbuzz.com/
Expires: Tue, 27-Nov-18 21:20:25 GMT; path=/; domain=.cricbuzz.com; HttpOnly
Etag:
number of documents in cache is less than 10
closing client connected on socket: 4
closing host associated with client connected on socket: 5
```

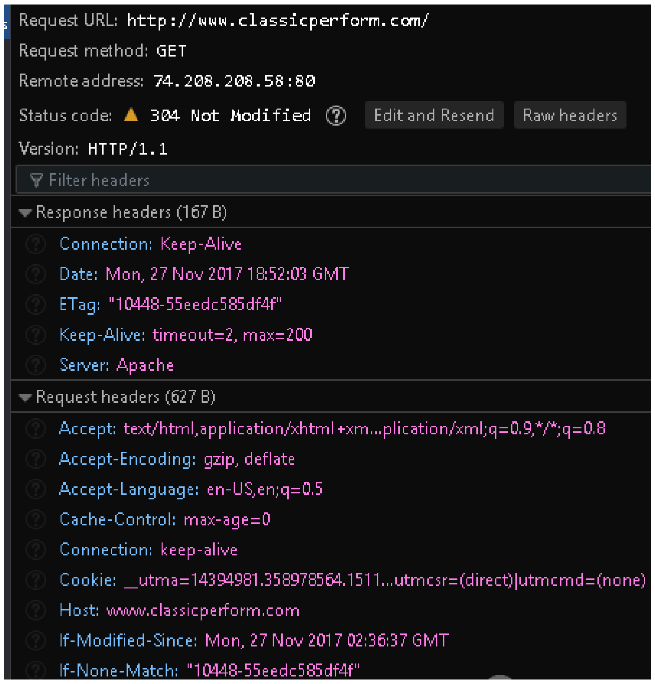
Requesting the same document again after a few seconds:



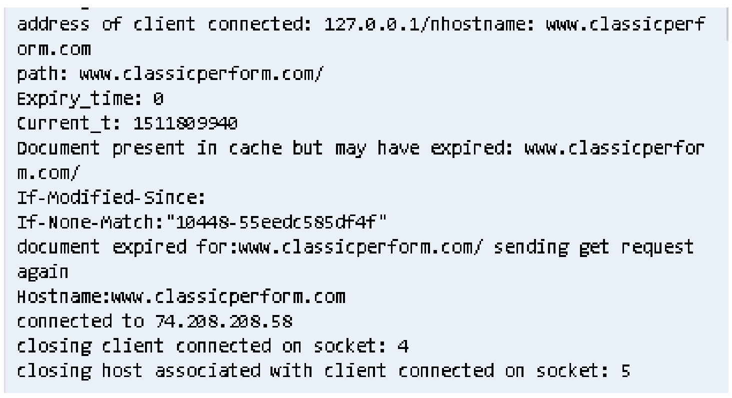
```
address of client connected: 127.0.0.1/nhostname: www.cricbuzz.com
path: www.cricbuzz.com/
Expiry_time: -61570550375
Current_t: 1511817742
Document present in cache but may have expired: www.cricbuzz.com/
deleting document form cache:www.cricbuzz.com/
document was not cached:www.cricbuzz.com/ requesting from the server
Hostname:www.cricbuzz.com
connected to 104.17.41.113
document cached:www.cricbuzz.com/
Expires: Tue, 27-Nov-18 21:22:22 GMT; path=/; domain=.cricbuzz.com; HttpOnly
Etag:
number of documents in cache is less than 10
closing client connected on socket: 4
closing host associated with client connected on socket: 5
```

We can see that the page has expired and hence has been deleted from the cache and a request has been sent to the server again.

5. A stale entry in the cache without an Expires header is determined based on the If-Modified-Since and If-None-Match field



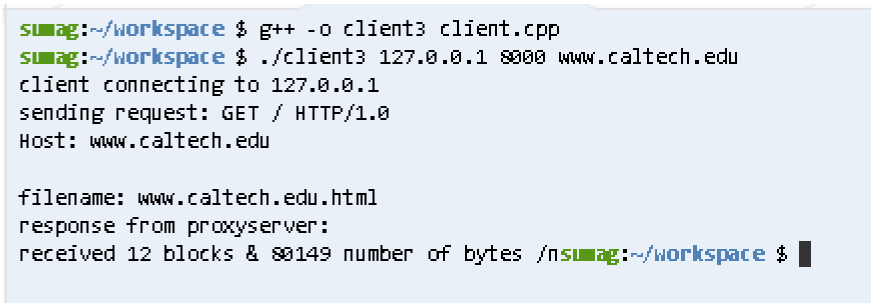
```
Request URL: http://www.classicperform.com/
Request method: GET
Remote address: 74.208.208.58:80
Status code: ▲ 304 Not Modified
Version: HTTP/1.1
Response headers (167 B)
  Connection: Keep-Alive
  Date: Mon, 27 Nov 2017 18:52:03 GMT
  ETag: "10448-55eedc585df4f"
  Keep-Alive: timeout=2, max=200
  Server: Apache
Request headers (627 B)
  Accept: text/html,application/xhtml+xm...plication/xml;q=0.9,*/*;q=0.8
  Accept-Encoding: gzip, deflate
  Accept-Language: en-US,en;q=0.5
  Cache-Control: max-age=0
  Connection: keep-alive
  Cookie: __utma=14394981.358978564.1511...utmcsr=(direct)|utmcmd=(none)
  Host: www.classicperform.com
  If-Modified-Since: Mon, 27 Nov 2017 02:36:37 GMT
  If-None-Match: "10448-55eedc585df4f"
```



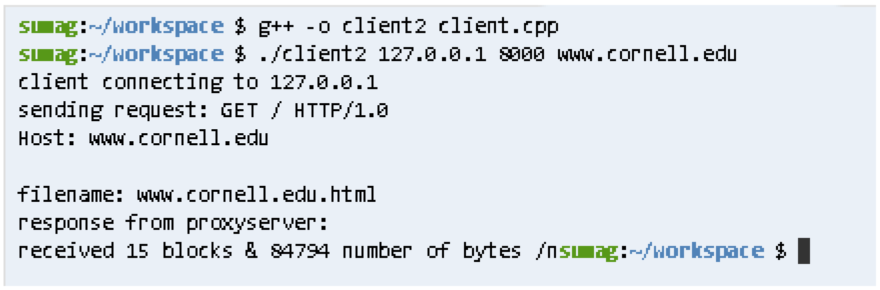
```
address of client connected: 127.0.0.1/nhostname: www.classicperf
orm.com
path: www.classicperform.com/
Expiry_time: 0
Current_t: 1511809940
Document present in cache but may have expired: www.classicperfor
m.com/
If-Modified-Since:
If-None-Match:"10448-55eedc585df4f"
document expired for:www.classicperform.com/ sending get request
again
Hostname:www.classicperform.com
connected to 74.208.208.58
closing client connected on socket: 4
closing host associated with client connected on socket: 5
```

We can clearly see that due to the absence of an expires timer, we have to use a conditional get request and check the values of if modified since and if none match which can be seen in the screenshot along with expiry_time and current_t.

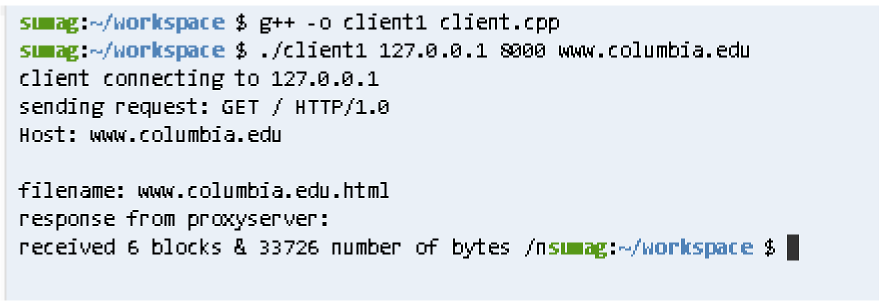
7. 3 clients simultaneously access the proxy server and get the correct data



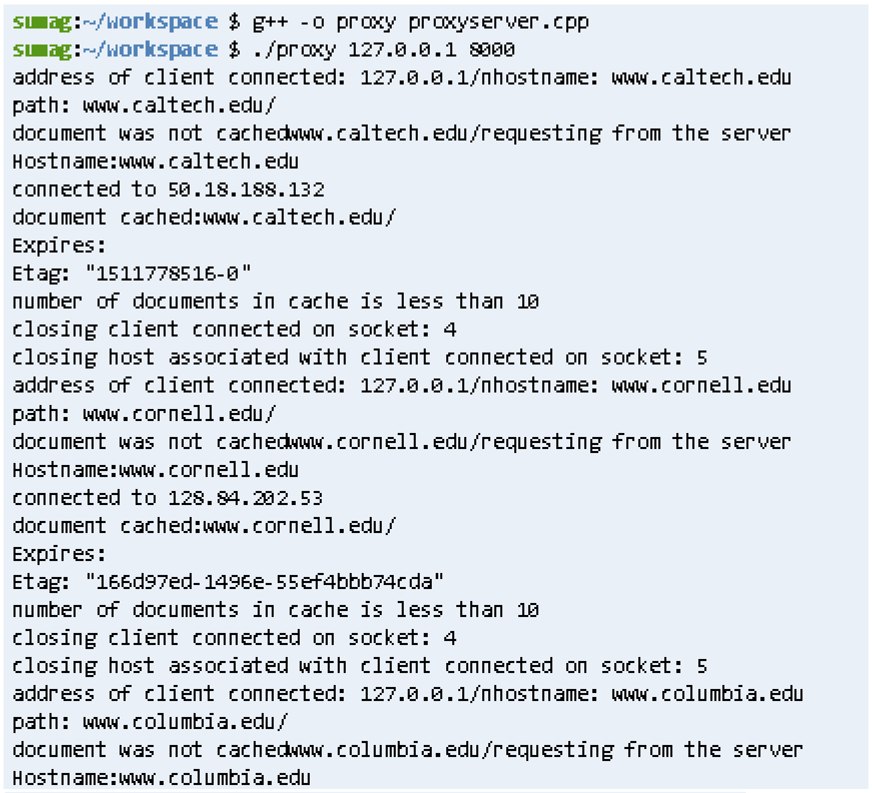
```
sumag:~/workspace $ g++ -o client3 client.cpp
sumag:~/workspace $ ./client3 127.0.0.1 8000 www.caltech.edu
client connecting to 127.0.0.1
sending request: GET / HTTP/1.0
Host: www.caltech.edu

filename: www.caltech.edu.html
response from proxyserver:
received 12 blocks & 80149 number of bytes /nsumag:~/workspace $ █
```



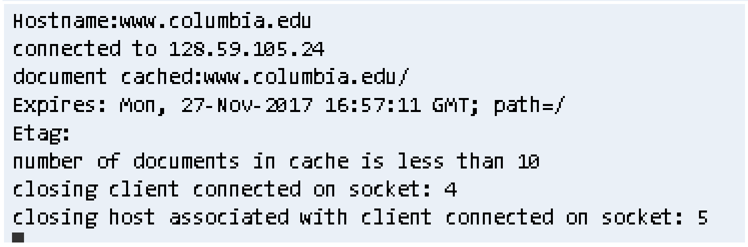
```
sumag:~/workspace $ g++ -o client2 client.cpp
sumag:~/workspace $ ./client2 127.0.0.1 8000 www.cornell.edu
client connecting to 127.0.0.1
sending request: GET / HTTP/1.0
Host: www.cornell.edu

filename: www.cornell.edu.html
response from proxyserver:
received 15 blocks & 84794 number of bytes /nsumag:~/workspace $ █
```



```
sumag:~/workspace $ g++ -o client1 client.cpp
sumag:~/workspace $ ./client1 127.0.0.1 8000 www.columbia.edu
client connecting to 127.0.0.1
sending request: GET / HTTP/1.0
Host: www.columbia.edu

filename: www.columbia.edu.html
response from proxyserver:
received 6 blocks & 33726 number of bytes /nsumag:~/workspace $ █
```



```
sumag:~/workspace $ g++ -o proxy proxyserver.cpp
sumag:~/workspace $ ./proxy 127.0.0.1 8000
address of client connected: 127.0.0.1/nhostname: www.caltech.edu
path: www.caltech.edu/
document was not cachedwww.caltech.edu/requesting from the server
Hostname:www.caltech.edu
connected to 50.18.188.132
document cached:www.caltech.edu/
Expires:
Etag: "1511778516-0"
number of documents in cache is less than 10
closing client connected on socket: 4
closing host associated with client connected on socket: 5
address of client connected: 127.0.0.1/nhostname: www.cornell.edu
path: www.cornell.edu/
document was not cachedwww.cornell.edu/requesting from the server
Hostname:www.cornell.edu
connected to 128.84.202.53
document cached:www.cornell.edu/
Expires:
Etag: "166d97ed-1496e-55ef4bbb74cda"
number of documents in cache is less than 10
closing client connected on socket: 4
closing host associated with client connected on socket: 5
address of client connected: 127.0.0.1/nhostname: www.columbia.edu
path: www.columbia.edu/
document was not cachedwww.columbia.edu/requesting from the server
Hostname:www.columbia.edu
```



```
Hostname:www.columbia.edu
connected to 128.59.105.24
document cached:www.columbia.edu/
Expires: Mon, 27-Nov-2017 16:57:11 GMT; path=/
Etag:
number of documents in cache is less than 10
closing client connected on socket: 4
closing host associated with client connected on socket: 5
```

Source Code

Proxyserver.cpp

```
#include<iostream>
#include<string.h>
#include<string>
#include<stdio.h> // for file related functions
#include<sys/socket.h> //sendto and recvfrom functions
#include<sys/select.h> //for select()
#include<stdlib.h>
#include<string.h> //memset
#include<sys/socket.h>// for socklen_t
#include<arpa/inet.h> //ntop
#include<unistd.h>
#include<inttypes.h>
#include<errno.h>
#include<netdb.h>
```

```
#include<assert.h>
#include<stdarg.h>

#include<vector>
#include<map>
#include<algorithm>
#include<list>
using namespace std;
//Structure to store requested url and socket information related to the client.
struct information_client
{
 string path;
 char hostname[1024];
 int client_fd;
 int host_fd;
};

vector<information_client*> list_client;
//Structure to define data blocks
struct block_data
{
 char data[10240];
 int data_size;
};
//Structure to store information related to a document.
struct document
{
 vector <block_data*> block;
 string path_name;
 char expires[255];
 char etag[255];
 time_t last_modified, expiry_time;
 bool check;
};
//Structure to store cache information where path of the document is mapped to the pointer to document.
struct cache
{
 map<string,document*> document_map;
 list<string> list_path;
};
//function to extract information from the headers.
bool info_extract(const char* key, char* buffer, char* extracted_info)
 {
  char *begin=strstr(buffer,key);
  if(!begin)
  {
   return false;
  }
  else
  {
```

```
  char* end=strstr(begin, "\r\n");
  begin= begin+ strlen(key);
  while(*begin==' ') ++begin;
  while(*(end-1)==' ') --end;
  strncpy(extracted_info, begin, end-begin);
  extracted_info[end-begin]='\0';
  //cout<<"Extracted Info is :"<< extracted_info<<endl;
  return true;
  }
 }
 //Function to extract the time and convert it into easily comparable format.
 time_t retrieve_time(const char* key, char* buffer)
 {
  char extracted_time[255]={0};
  bool x= info_extract(key, buffer, extracted_time);
    if(!x)
  {
   return 0;
  }
  struct tm t={0};
  char* t_ptr=strptime(extracted_time, "%a, %d-%b-%Y %H:%M:%S %Z", &t);
  if(!t_ptr) return 0;
  else return mktime(&t);
 }
//Fucntion to get the current time of the system.
time_t Gettingtime()
 {
  time_t raw_t;
  time(&raw_t);
  struct tm *pointer_t;
  pointer_t=gmtime(&raw_t);
  return mktime(pointer_t);
 }
 //Function to retrieve etag and expires field information
 void retrieve_time_info (char* buffer, const int numbytes, document* document_pointer)
 {
  time_t expires_t, lastmodified_t;
  expires_t= retrieve_time("expires=", buffer) ;
  if(!expires_t)
  {
   expires_t= retrieve_time("expires:", buffer) ;
  }
  lastmodified_t=retrieve_time("Last-Modified:", buffer);
  if(expires_t)
  {
   document_pointer->expiry_time=expires_t;
  }
  if(lastmodified_t)
  {
   document_pointer->last_modified=lastmodified_t;
  }
  if(info_extract("Etag: ", buffer, document_pointer->etag)==false)
  {
   info_extract("ETag: ", buffer, document_pointer->etag);
  }
  info_extract("expires=", buffer, document_pointer->expires);
  cout<< "Expires: "<< document_pointer->expires<<endl;
  cout<<"Etag: "<<document_pointer->etag<<endl;
 }
 //To delete blocks of data when cache is full or the entry is not current.
 void deleting_block(document* document_pointer)
 {
  vector<block_data*>::iterator it;
  for (it=document_pointer->block.begin();it!=document_pointer->block.end();it++)
  {
   delete *it;
  }
  document_pointer->block.clear();
 }
 //To add a new document to cache when it is not present in the cache already.
  bool addingdocument(document* document_pointer, cache* proxy_cache)
 {
  proxy_cache->list_path.push_front(document_pointer->path_name);
  proxy_cache->document_map[document_pointer->path_name]=document_pointer;

  if(proxy_cache->list_path.size() <= 10)
  {
   cout << "number of documents in cache is less than 10" <<endl;
   return false;
  }
  else
  {
  string path_name= proxy_cache->list_path.back();
  cout<< "cache size exceeded, deleting path:" << path_name << endl;
  deleting_block(proxy_cache->document_map[path_name]);
  delete proxy_cache->document_map[path_name];
  proxy_cache->document_map.erase(path_name);
  proxy_cache->list_path.pop_back();
  return true;
  }
 }
 //Retrieving the document when the path is known and is present in the cache.
 document* getting_document(string path_name, cache* proxy_cache)
{
 if(proxy_cache->document_map.find(path_name)==proxy_cache->document_map.end())
 return NULL;
 proxy_cache->list_path.splice(proxy_cache->list_path.begin(), proxy_cache->list_path, find(proxy_cache-
>list_path.begin(),proxy_cache->list_path.end(), path_name));
 return proxy_cache->document_map[path_name];
}
```

```
//To delete documents from cache when it is full or the entry is not current.
bool deleting_document(string path_name, cache *proxy_cache)
 {
  deleting_block(proxy_cache->document_map[path_name]);
  delete proxy_cache->document_map[path_name];
  proxy_cache->document_map.erase(path_name);
  proxy_cache->list_path.erase(find(proxy_cache->list_path.begin(),proxy_cache->list_path.end(), path_name));
  return true;
 }
 //Finding the client information among multiple clients when receivin a request from it.
 vector<information_client*>::iterator findingclient(int sock_fd)
{
 vector<information_client*>::iterator it;
 for (it=list_client.begin(); it!=list_client.end(); it++)
 {
  if ((*it)->client_fd==sock_fd || (*it)->host_fd==sock_fd)
  return it;
 }
 return list_client.end();
}
//for IPv4 and IPv6
 void *get_in_addr (struct sockaddr *sa)
{
  if (sa -> sa_family == AF_INET)
  {
   return &(((struct sockaddr_in*)sa)->sin_addr);
  }
   return &(((struct sockaddr_in6*)sa)->sin6_addr);
 }
 //To close a connection to client/host on not receiving information when requested(idle/unresponsive).
 int connection_close(information_client* client, fd_set& master_fd)
 {
  if (client->client_fd!=-1)
  {
   close(client->client_fd);
   FD_CLR(client->client_fd, &master_fd);
   printf("closing client connected on socket: %d \n", client->client_fd);
  }
  if(client->host_fd!=-1)
   {
   close(client->host_fd);
   FD_CLR(client->host_fd, &master_fd);
   printf("closing host associated with client connected on socket: %d \n", client->host_fd);
  }
  delete client;
  return 0;
 }
 //To find the hostname and determine the path of the document.
  int path_info(const char* buffer, string& path, char* hostname)
 {
```

```
 char buf_response[1024];
 strcpy(buf_response, buffer);
 char* host_start= strstr(buf_response, "Host: ");
 host_start = host_start + strlen("Host: ");
 char* host_end= strchr(host_start, ' ');

 if(!host_end)
  {
  host_end= strchr(host_start, '\r');
  }

  strncpy(hostname, host_start, host_end-host_start);
  hostname[host_end-host_start]='\0';

  path.append(hostname);
  char* ptr_path;
  ptr_path= strtok(buf_response, " \r\n");

  if (ptr_path!=NULL)
  {
   ptr_path=strtok(NULL," \r\n" );
   path.append(ptr_path);
  }
  return 0;
}
//Establishing a connection to the host.
int connectingtohost(char* host)
{
 char http_host[INET6_ADDRSTRLEN];
 int sockfd, rv;
 struct addrinfo hints, *servinfo, *p;

 memset(&hints, 0, sizeof hints);
 hints.ai_family=AF_UNSPEC;
 hints.ai_socktype=SOCK_STREAM;

 if ((rv=getaddrinfo(host,"80",&hints,&servinfo))!=0)
 {
  fprintf(stderr, "getaddrinfo: %s \n", gai_strerror(rv));
  return 1;
 }

 for(p=servinfo;p!=NULL;p=p->ai_next)
 {
  if ((sockfd=socket(p->ai_family, p->ai_socktype, p->ai_protocol))==-1)
  {
   perror("socket error");
   continue;
  }
  if(connect(sockfd,p->ai_addr, p->ai_addrlen)==-1)
```

```
   {
    close(sockfd);
    perror("connect error");
    continue;
   }
   break;
  }
 if (p==NULL)
 {
  fprintf (stderr, "connection to host failed\n");
  return -1;
 }
 inet_ntop(p->ai_family, get_in_addr((struct sockaddr*)p->ai_addr), http_host, sizeof http_host);
 printf("connected to %s \n", http_host);
 freeaddrinfo(servinfo);
 return sockfd;
}
cache *proxy_cache= new cache;
//Updating the cache with the requested document and related info based on request type(200 or 304)
void cache_update(information_client* client, char* buffer, int numbytes)
{
 document* document_pointer=getting_document(client->path, proxy_cache);
 if(!document_pointer && (strstr(buffer,"HTTP/1.0 200 OK") || strstr(buffer,"HTTP/1.1 200 OK")))
 {
  document_pointer=new document;
  memset(document_pointer->expires,0,255);
  memset(document_pointer->etag,0,255);
  document_pointer->last_modified=0;
  document_pointer->expiry_time=0;
  document_pointer->check=false;
  document_pointer->path_name=client->path;

  cout<<"document cached:" << client->path <<endl;
  retrieve_time_info(buffer,numbytes,document_pointer);
  addingdocument (document_pointer, proxy_cache);
 }// checking for Conditional GET request situation
 if(document_pointer->check)
 {
  //If the buffer contains header information extract the information.
  if(strstr(buffer, "HTTP/1.0 304 Not Modified") || strstr(buffer, "HTTP/1.1 304 Not Modified"))
  {
   cout<< "HTTP/1.0 304 Not Modified" <<endl;
   retrieve_time_info(buffer, numbytes, document_pointer);
   return;
  }

  else if (strstr(buffer, "HTTP/1.0 200 OK") || strstr(buffer, "HTTP/1.1 200 OK"))
  {
   cout<< "HTTP/1.0 200 OK" <<endl;
   retrieve_time_info(buffer, numbytes, document_pointer);
```

```
    deleting_block(document_pointer);
   }
   document_pointer->check=false;
  }

 block_data* blkptr = new block_data;
 memcpy(blkptr->data,buffer, numbytes);
 blkptr->data_size=numbytes;
 document_pointer->block.push_back(blkptr);
 }
//To check for the If-Modified-Since field to check if conditional GET is required or not.
bool cache_check(document* document_pointer, char* buffer, int& numbytes)
{
 char* ending= strstr(buffer, "\r\n\r\n");
 if(!ending || document_pointer->expiry_time|| !strlen(document_pointer->etag))
 {
  return false;
 }
 else
 {
 ending = ending+2;
 numbytes= numbytes+ sprintf(ending, "%s%s\r\n%s%s\r\n\r\n", "If-Modified-Since", document_pointer->expires, "If-
None-Match", document_pointer->etag);
 info_extract("If-Modified-Since:", buffer, document_pointer->expires);
 cout<< "If-Modified-Since:" << document_pointer->expires <<endl;
 cout<< "If-None-Match:"<< document_pointer->etag<<endl;
 numbytes=numbytes-2;
 return true;
 }
}

//vector<information_client*> list_client;
//cache *proxy_cache= new cache;

 int main(int argc, char *argv[])
 {
  struct addrinfo hints, *servinfo, *p;
  int fdmax;
  int listener; //listening for new connections
  fd_set master_fd;  //master set to store the file descriptors of multiple clients.
  fd_set read_fds;
  int rv;
  int optval=1;

  if (argc!=3)
  {
   printf("usage: /proxy <ip to bind> <port to bind> \n");
   return -1;
  }
```

```
FD_ZERO(&master_fd);
FD_ZERO(&read_fds);

memset(&hints, 0, sizeof hints);
hints.ai_family= AF_UNSPEC;
hints.ai_socktype= SOCK_STREAM;
hints.ai_flags= AI_PASSIVE;

if ((rv=getaddrinfo(argv[1],argv[2],&hints,&servinfo))!=0)
{
 fprintf(stderr, "proxyserver %s \n", gai_strerror(rv));
 exit(1);
}

for (p=servinfo; p!=NULL; p=p->ai_next)
{

 listener= socket (p->ai_family, p->ai_socktype, p->ai_protocol);
 if (listener<0) continue;
 setsockopt(listener, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof (int));

 if (bind(listener, p->ai_addr, p->ai_addrlen)<0)
 {
  close(listener);
  continue;
 }
 break;
}

if (p==NULL)
{
 fprintf (stderr, "proxyserver: failed to bind \n");
 exit(2);
}

freeaddrinfo(servinfo);

if (listen(listener, 10)==-1)
{
 perror("listen");
 exit(3);
}
//adding the current file descriptor to master set
FD_SET(listener, &master_fd);
fdmax= listener;

for(;;)
{
 read_fds=master_fd;
 if(select(fdmax+1, &read_fds, NULL, NULL, NULL)==-1)
```

```
  {
   perror("select");
   exit(4);
  }
  //loop through all connections to check if we are receiving data
  for(int i=1; i<=fdmax; i++)
  {
   if (FD_ISSET(i,&read_fds))
   {
    if(i==listener) //listening from new connection
    {
     struct sockaddr_storage client_addr;
     socklen_t address_length;
     address_length=sizeof client_addr;
     int ephemeral_fd;
     char client_ip[INET6_ADDRSTRLEN];

     if((ephemeral_fd=accept(listener,(struct sockaddr *)&client_addr, &address_length))==-1) //accept connection on the
ephemeral port of proxy server
     {
      perror("accept");
     }
     else
     {
      FD_SET(ephemeral_fd,&master_fd);
      if (ephemeral_fd>fdmax)
      {
       fdmax=ephemeral_fd; //updating the maximum value of the file descriptor to new value.
      }
      information_client* client=new information_client;
      client->host_fd=-1;
      client->client_fd=ephemeral_fd;
      list_client.push_back(client);
      printf("address of client connected: %s/n", inet_ntop(client_addr.ss_family, get_in_addr((struct sockaddr
*)&client_addr),client_ip, INET6_ADDRSTRLEN));
     }
    }
    else //listening from existing client or web server
    {
     char buffer[10240];
     int numbytes;
     if((numbytes=recv(i, buffer, sizeof buffer, 0))<=0) //sending data to buffer from i
     {
      if(numbytes<0)
      {
       perror("recv error");
      }
      vector<information_client*>::iterator it=findingclient(i);
      connection_close(*it, master_fd);
      list_client.erase(it);
     }
     else
     {
      information_client* client=*(findingclient(i));
      //received request from client
      if (i==client->client_fd)
      {
       bool check= false, retrieve_new=true ;
       path_info(buffer, client->path, client->hostname);
       cout<<"hostname: "<<client->hostname<<endl;
       cout<<"path: "<<client->path<<endl;
       document* document_pointer = getting_document(client->path, proxy_cache);
       if(document_pointer)//this document present in cache
       {
        time_t current_t= Gettingtime();
        cout<<"Expiry_time: " <<document_pointer->expiry_time<<endl;
        cout<<"Current_t: " << current_t<<endl;
        if(document_pointer->expiry_time>current_t) //retieving an unexpired document from the cache
        {
         retrieve_new=false;
         cout<<" Document already present in cache and is current: "<< client->path<<endl;
         vector <block_data*>:: iterator it;
         for (it=document_pointer->block.begin();it!=document_pointer->block.end();it++)
         {
          int send_data=send(client->client_fd, (*it)->data, (*it)->data_size, 0);
          if (send_data==-1) perror ("sending error");
         }
         vector<information_client*>::iterator it_1= findingclient(i);
         connection_close(*it_1, master_fd);
         list_client.erase(it_1);
        }
        else
        {
         cout<<"Document present in cache but may have expired: "<<client->path<<endl;
         check= cache_check(document_pointer, buffer, numbytes);
         if(check)
         {
         document_pointer->check=true;
         }
         else
         {
          cout<<"deleting document form cache:"<<client->path<<endl;
          deleting_document(client->path, proxy_cache);
         }

        }
       }
       if (retrieve_new) //document expired, hence requesting from server or document was not cahed, therefore requesting
from server.
       {
        if(check)
        {
         cout<<"document expired for:" << client->path<<" sending get request again" <<endl;
        }
        else
        cout<<"document was not cached:"<<client->path<<" requesting from the server"<<endl;
       if(client->host_fd==-1)
       {
        cout<<"Hostname:"<<client->hostname<<endl;
        client->host_fd = (connectingtohost(client->hostname)); //connecting to the web server to retrieve information
        FD_SET(client->host_fd, &master_fd);
        if(client->host_fd>fdmax) fdmax=client->host_fd;
       }
       if(send(client->host_fd, buffer, numbytes, 0)==-1) perror("sending error");
       }
      }
      //received response from server
      else if (i==client->host_fd)
      {
       cache_update(client, buffer, numbytes);
       document* document_pointer= getting_document(client->path, proxy_cache);
       if(document_pointer && (strstr(buffer, "HTTP/1.0 304 Not Modified") || strstr(buffer, "HTTP/1.0 304 Not
Modified")))
       {
        vector <block_data*>::iterator it;
        for(it=document_pointer->block.begin();it!=document_pointer->block.end();it++)
        {
         if (send(client->client_fd, (*it)->data,(*it)->data_size, 0) == -1) perror("sending error");
        }
         cout<< "requested data sent from cache: "<<client->path<<endl;
         vector <information_client*> ::iterator it_1=findingclient(i);
         connection_close(*it_1, master_fd);
         list_client.erase (it_1);
        }
        else if (send(client->client_fd, buffer, numbytes, 0)==-1)
        {
         perror("sending error");
        }
       }
      }
     }
    }
   }
  }
 return 0;
}
```

Client.cpp

```
#include<iostream>
```

```
#include<string.h>
#include<string>
#include<stdio.h> // for file related functions
#include<sys/socket.h> //sendto and recvfrom functions
#include<sys/select.h> //for select()
#include<stdlib.h>
#include<string.h> //memset
#include<sys/socket.h>// for socklen_t
#include<arpa/inet.h> //ntop
#include<unistd.h>
#include<inttypes.h>
#include<errno.h>
#include<netdb.h>
#include<assert.h>
#include<stdarg.h>
using namespace std;
/*function to extract the hostname and filename from url and
send the GET request to the proxy server*/
int request(char* url,char *buffer, char *file_name)
{
    int length;
    char* start= strstr(url, "http://");
    if (start)
    {
       url=url+strlen("http://");
    }
        char* end= strchr (url, '/');

    if (end)
    {
        sprintf (file_name, "%s", end+1);
        char hostname[1024];
        strncpy(hostname, url, end-url);
        hostname[end-url]='\0';
        printf("sending request: GET %s HTTP/1.0\r\nHost: %s\r\n\r\n", end, hostname);
                          length=sprintf(buffer, "GET %s HTTP/1.0\r\n Host: %s\r\n\r\n", end, hostname);
    }
    else
     {
         sprintf (file_name, "%s", url);
         strcat(file_name, ".html");
         printf("sending request: GET / HTTP/1.0\r\nHost: %s\r\n\r\n", url);
                   length=sprintf(buffer, "GET / HTTP/1.0\r\nHost: %s\r\n\r\n", url);
     }

     return 1+length; //returns the number of characters written to the buffer.
}

//for IPv4 and IPv6
void *get_in_addr (struct sockaddr *sa)
```

```
{
 if (sa -> sa_family == AF_INET)
 {
  return &(((struct sockaddr_in*)sa)->sin_addr);
 }
  return &(((struct sockaddr_in6*)sa)->sin6_addr);
 }

 int main (int argc, char *argv[])
 {
 int sockfd, numbytes, rv;
 char buffer_request[1024];
 char buffer_response[10240];
 char file_name[50];
 char url[1024];
 char proxyserver[INET6_ADDRSTRLEN];
 struct addrinfo hints, *servinfo, *p;

 if (argc!=4)
 {
 fprintf (stderr, "usage: ./client <proxy address> <proxy port> <URL to retrieve>");
 exit(1);
 }

 strcpy(url, argv[3]);

 memset (&hints, 0, sizeof hints);
 hints.ai_family = AF_UNSPEC;
 hints.ai_socktype=SOCK_STREAM;

 if ((rv=getaddrinfo (argv[1], argv[2], &hints, &servinfo))!=0)
 {
 fprintf (stderr, "getaddrinfo: %s\n", gai_strerror (rv));
 return 1;
 }

 for (p=servinfo; p!=NULL; p=p->ai_next)
 {
 if ((sockfd=socket(p->ai_family, p->ai_socktype, p->ai_protocol))==-1)
 {
 perror ("client:socket");
 continue;
 }
 //establishing the connection
 if (connect(sockfd, p->ai_addr, p->ai_addrlen)==-1)
 {
 close(sockfd);
 perror("client:connect");
 continue;
 }
 break;
 }

 if (p==NULL)
 {
 fprintf(stderr, "client: failed to connect \n");
 return 2;
 }

 inet_ntop (p->ai_family, get_in_addr((struct sockaddr *)p->ai_addr),proxyserver, sizeof proxyserver);
 printf ("client connecting to %s \n", proxyserver);
 freeaddrinfo(servinfo);

 memset(file_name, 0, 50);

 numbytes=request(url, buffer_request, file_name);
 printf ("filename: %s\n", file_name);
 FILE *ostream;

 if (*file_name)
 {
     //Opening a new output file to store the received information from the proxy server.
     if ((ostream=fopen(file_name, "wb"))==NULL)
     {
         perror("error in opening file");
         printf("error: %d (%s) \n", errno, strerror(errno));
         return -1;
     }
 }
 //Sending GET request
 if (send(sockfd, buffer_request, numbytes, 0)==-1)
  {
     perror("send error");
  }

 printf ("response from proxyserver: \n");
 int block_num=0;
 size_t bytes_num=0;
 int block_1=1;
 char* ptr_block_1;

//Receiving data block by block and writing it to ostream while receiving some data from the proxy server.
do
 {
     if ((numbytes= recv(sockfd, buffer_response, 10239,0))==-1)
     {
         perror("recv error");
         exit(1);
     }
```

```
    buffer_response[numbytes]='\0';
    bytes_num= bytes_num + numbytes;
    block_num++;

    if (*file_name)
    {
        if (block_1)
        {
            ptr_block_1 = strstr (buffer_response, "/r/n/r/n");
            if (ptr_block_1) ptr_block_1 = ptr_block_1 + strlen("/r/n/r/n");
            fwrite(ptr_block_1, 1,numbytes-(ptr_block_1-buffer_response), ostream);
            block_1=0;
        }

        else
        fwrite(buffer_response, 1, numbytes, ostream);
    }
} while (numbytes);

if (*file_name)
{
    fclose(ostream);
}

close(sockfd);
//Closing the socket.
printf("received %d blocks & %lu number of bytes /n", block_num, bytes_num);
return 0;
}
```

Acknowledgements

1. Beej’s Guide
2. www.cplusplus.com

Multiple references from GeeksforGeeks, Stackoverflow.