**Programming Assignment 4**

Chenjie Luo (UIN: 324007289)
Yu-Wen Chen(UIN: 227009499)

## a. Description

The project is divided into two parts: server and client. In this assignment Chenjie Luo is responsible for server side as well as LRU cache while Yu-Wen Chen is responsible is responsible for client side and expire feature. We verified functionalities of two parts in different test cases together.

For the server's architecture, when target file is executed, users will type in two inputs which are: server's IP address as well as port number. After creating the socket and bind successfully, we will listen to all sockets in FD_SET. When a request is received, we will firstly check if the url is cached in our LRU cache and not expire yet. If that is the case, we will directly respond to the client using the data in cache. Otherwise, we will need to send request to the target web server and request from it. The read buffer size I designed for url is 1024 bytes from client and the buffer to transmit html data to 102400 bytes. After we received it, we added it into cache and respond to the client as well. To realize this functionality, I designed and implemented our LRU cache. Firstly, I designed a data structure called Node to store entities in the LRU cache. It consists of key (url), data (html data), expireat, next (next Node) and last_modified. For the LRU cache, it basically removed least recently used data from cache. Therefore, a linked-list-like structure could achieve this functionality. But since we will need to realize the functionality to move certain node to the tail of linked list when it was accessed, I used unordered_map to implement the cache. For each element in unordered_map, it mapped key(url) to its previous node and then this could help me access the address of current node which are to be moved to the end of linked list. I designed implement LRUcache as an object. It contains several variables including current size of cache, capacity, Header (header node of the linked list), Tail (tail node of the linked list) and several member functions including get(), push() print_status() and moveToTail().

For the client side, the user needs to type server's IP address, server's port number and the requested URL as inputs. After receiving the URL, the client parses the URL into three portions: host name, data path and file name. Then, the client uses the input address and port number to connect to the server, sends the requested URL to the server and then waits for the server to provide the file from the requested URL. After receiving the file, the client first parses the first line of the file to get the response code and print it out, and then write the file into local directory with the parsed file name.

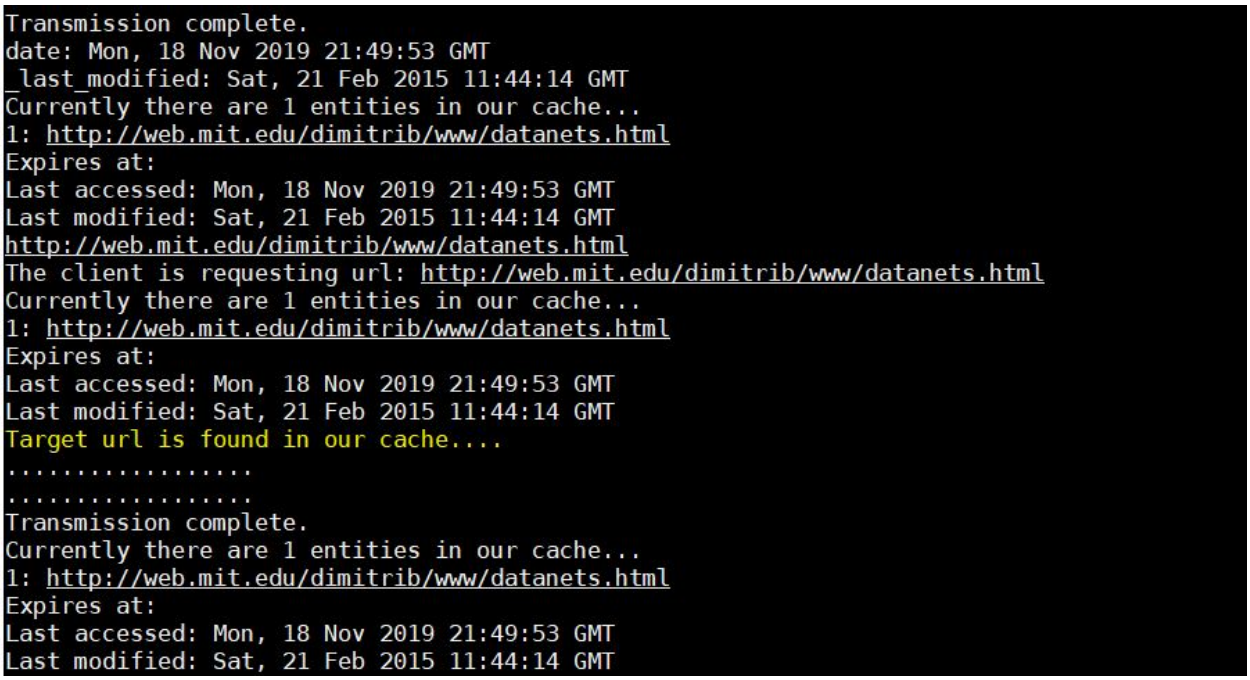
## b. Instruction to run our code

1. After downloading the file from github, type "make" in the command line to generate the execution file: server and client
2. Type "./server [server IP address] [port number]" to execute the server, and then type ./client [server IP address] [port number] [requested URL] to send request to server
3. Now, users can use the proxy server and client to get the file from the requested URL

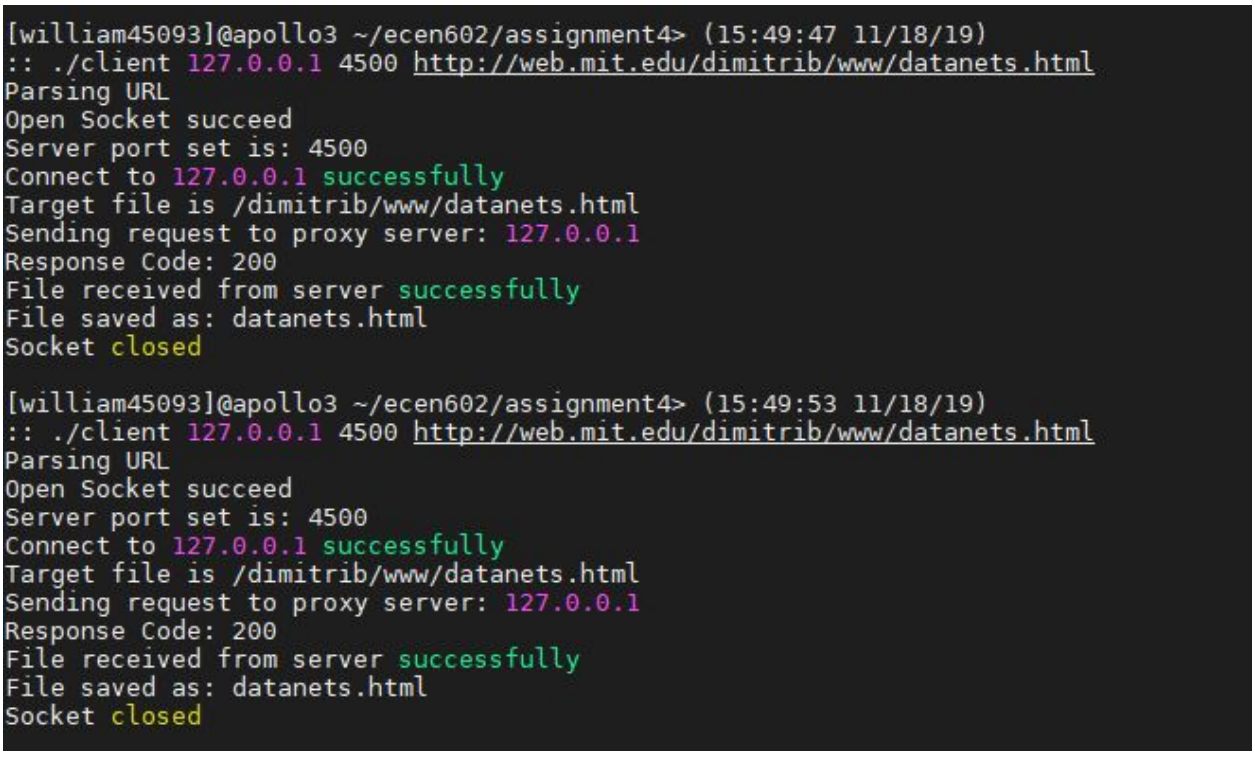
**c. Test result**

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

Server

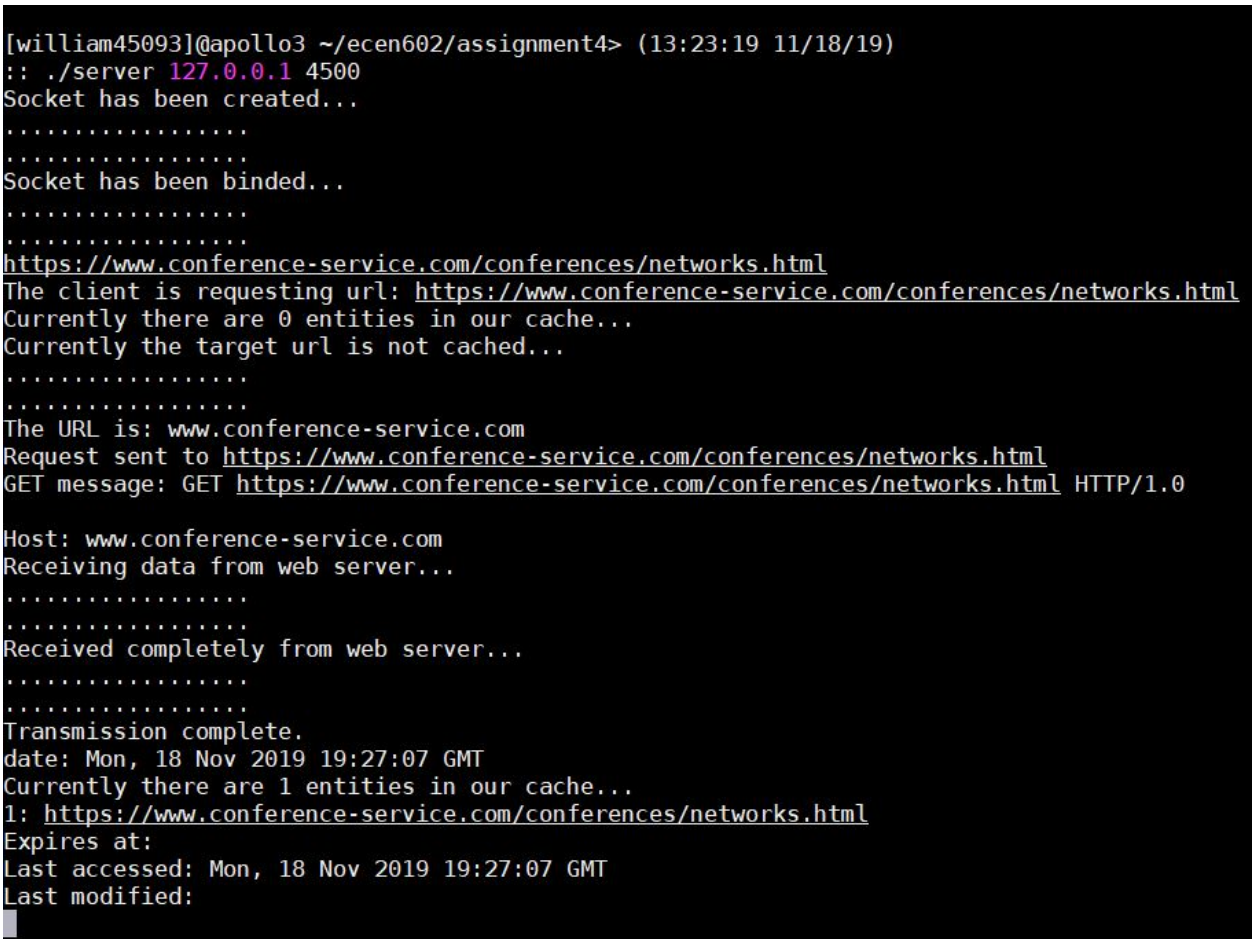



Client

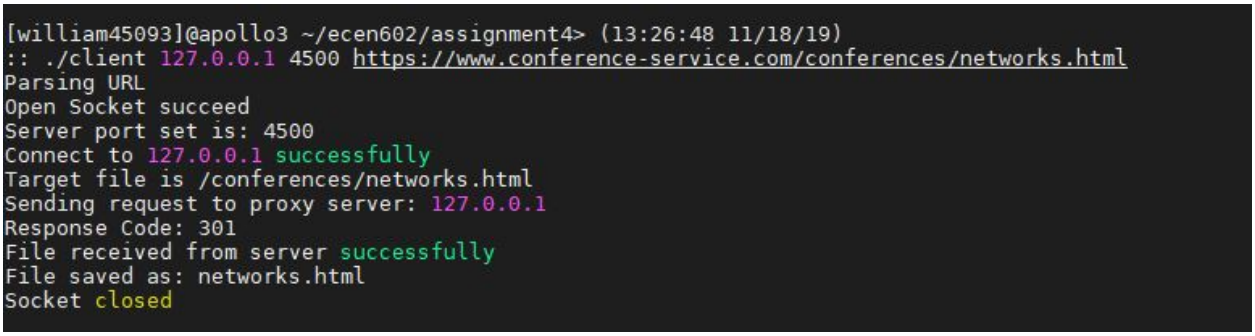


2. A request that is not in the cache is proxied, saved in the cache, and returned to the requester

Server




Client



3. A cache miss with 10 items already in the cache is proxied, saved in the LRU location in cache, and the data is returned to the requester
   Server
   Full cache

```
.................
Transmission complete.
date: Mon, 18 Nov 2019 19:55:10 GMT
Currently there are 10 entities in our cache...
1: https://www.conference-service.com/conferences/networks.html
Expires at:
Last accessed: Mon, 18 Nov 2019 19:52:51 GMT
Last modified:
2: www.codeincodeblock.com/2011/06/mini-project-snake-game-in-c.html
Expires at: Mon, 01 Jan 1990 00:00:00 GMT
Last accessed: Mon, 18 Nov 2019 19:51:49 GMT
Last modified:
3: http://mysmallwebpage.com/about-me.html
Expires at: Thu, 01 Jan 1970 00:00:00 UTC
Last accessed: Mon, 18 Nov 2019 19:53:51 UTC
Last modified:
4: www.evanjones.ca/crc32c.html
Expires at:
Last accessed: Mon, 18 Nov 2019 19:52:22 GMT
Last modified:
5: http://courses.cs.tamu.edu/teresa/csce221/csce221-index.html
Expires at:
Last accessed: Mon, 18 Nov 2019 19:50:03 GMT
Last modified: Fri, 28 Jun 2019 15:56:10 GMT
6: https://cesg.tamu.edu/research/
Expires at:
Last accessed: Mon, 18 Nov 2019 19:52:37 GMT
Last modified:
7: https://www.mcs.anl.gov/~kazutomo/rdtsc.html
Expires at:
Last accessed: Mon, 18 Nov 2019 19:49:58 GMT
Last modified:
8: http://ece.tamu.edu/~sunilkhatri/courses/ee449.html
Expires at:
Last accessed: Mon, 18 Nov 2019 19:55:10 GMT
Last modified:
9: www.cs.cornell.edu/~tomf/notes/cps104/twoscomp.html
Expires at:
Last accessed: Mon, 18 Nov 2019 19:50:20 GMT
Last modified:
10: http://faculty.cs.tamu.edu/schaefer/teaching/221_Fall2018/labs.html
Expires at:
Last accessed: Mon, 18 Nov 2019 19:49:55 GMT
Last modified: Fri, 24 Aug 2018 17:11:49 GMT
```

LRU cache: http://faculty.cs.tamu.edu/schaefer/teaching/221_Fall2018/labs.html is replaced

```
Transmission complete.
date: Mon, 18 Nov 2019 19:58:54 GMT
_last_modified: Mon, 14 Jan 2019 19:55:24 GMT
Currently there are 10 entities in our cache...
1: http://ece.tamu.edu/~sunilkhatri/courses/ee449.html
Expires at:
Last accessed: Mon, 18 Nov 2019 19:55:10 GMT
Last modified:
2: https://www.mcs.anl.gov/~kazutomo/rdtsc.html
Expires at:
Last accessed: Mon, 18 Nov 2019 19:49:58 GMT
Last modified:
3: https://cesg.tamu.edu/research/
Expires at:
Last accessed: Mon, 18 Nov 2019 19:52:37 GMT
Last modified:
4: http://courses.cs.tamu.edu/teresa/csce221/csce221-index.html
Expires at:
Last accessed: Mon, 18 Nov 2019 19:50:03 GMT
Last modified: Fri, 28 Jun 2019 15:56:10 GMT
5: www.cs.cornell.edu/~tomf/notes/cps104/twoscomp.html
Expires at:
Last accessed: Mon, 18 Nov 2019 19:50:20 GMT
Last modified:
6: www.evanjones.ca/crc32c.html
Expires at:
Last accessed: Mon, 18 Nov 2019 19:52:22 GMT
Last modified:
7: http://mysmallwebpage.com/about-me.html
Expires at: Thu, 01 Jan 1970 00:00:00 UTC
Last accessed: Mon, 18 Nov 2019 19:53:51 UTC
Last modified:
8: www.codeincodeblock.com/2011/06/mini-project-snake-game-in-c.html
Expires at: Mon, 01 Jan 1990 00:00:00 GMT
Last accessed: Mon, 18 Nov 2019 19:51:49 GMT
Last modified:
9: http://ece.tamu.edu/~xizhang/ECEN619/
Expires at:
Last accessed: Mon, 18 Nov 2019 19:58:54 GMT
Last modified: Mon, 14 Jan 2019 19:55:24 GMT
10: https://www.conference-service.com/conferences/networks.html
Expires at:
Last accessed: Mon, 18 Nov 2019 19:52:51 GMT
Last modified:
```

Client - first input cache is:
http://faculty.cs.tamu.edu/schaefer/teaching/221_Fall2018/labs.html

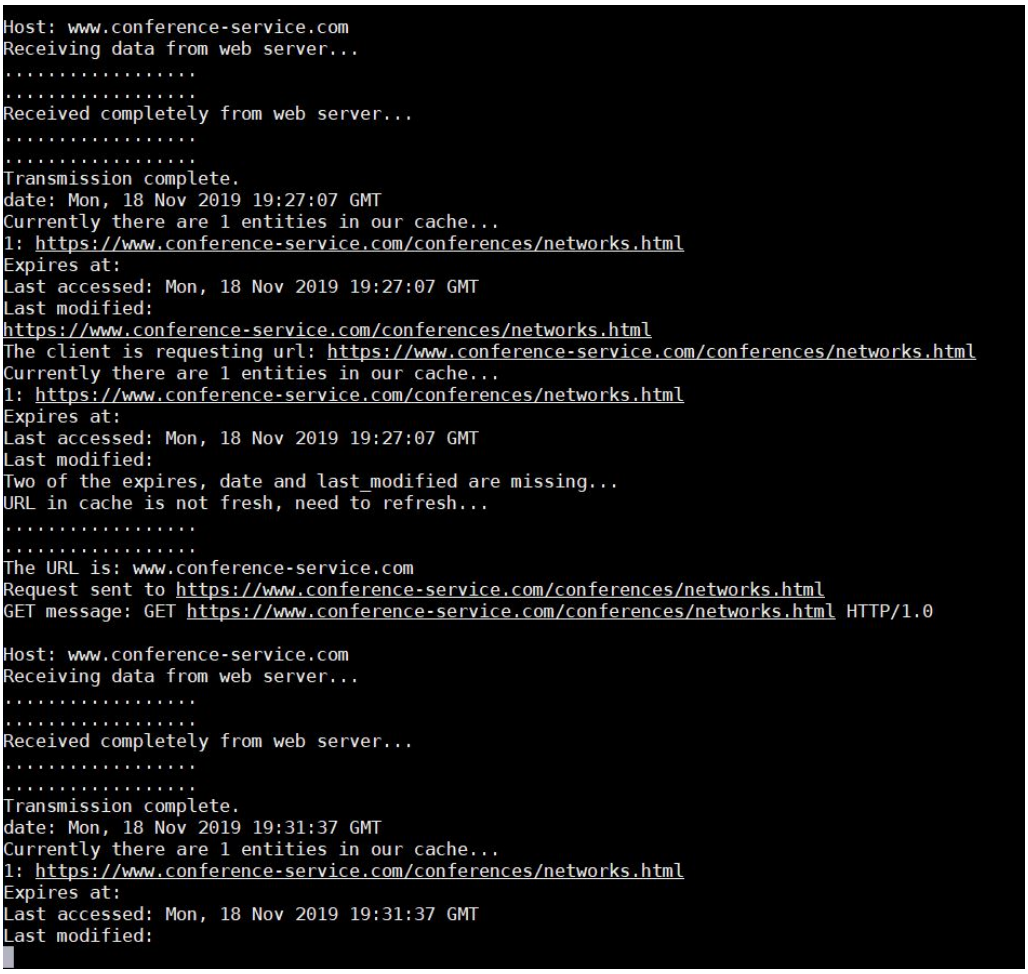
```
:: ./client 127.0.0.1 4500 http://faculty.cs.tamu.edu/schaefer/teaching/221_Fall2018/labs.html
Parsing URL
Open Socket succeed
Server port set is: 4500
Connect to 127.0.0.1 successfully
Target file is /schaefer/teaching/221_Fall2018/labs.html
Sending request to proxy server: 127.0.0.1
Response Code: 200
File received from server successfully
File saved as: labs.html
Socket closed

[william45093]@apollo3 ~/ecen602/assignment4> (13:49:55 11/18/19)
:: ./client 127.0.0.1 4500 https://www.mcs.anl.gov/~kazutomo/rdtsc.html
Parsing URL
Open Socket succeed
Server port set is: 4500
Connect to 127.0.0.1 successfully
Target file is /~kazutomo/rdtsc.html
Sending request to proxy server: 127.0.0.1
Response Code: 301
File received from server successfully
File saved as: rdtsc.html
Socket closed

[william45093]@apollo3 ~/ecen602/assignment4> (13:49:58 11/18/19)
:: ./client 127.0.0.1 4500 http://courses.cs.tamu.edu/teresa/csce221/csce221-index.html
Parsing URL
Open Socket succeed
Server port set is: 4500
Connect to 127.0.0.1 successfully
Target file is /teresa/csce221/csce221-index.html
Sending request to proxy server: 127.0.0.1
Response Code: 200
File received from server successfully
File saved as: csce221-index.html
Socket closed

[william45093]@apollo3 ~/ecen602/assignment4> (13:50:03 11/18/19)
:: ./client 127.0.0.1 4500 www.cs.cornell.edu/~tomf/notes/cps104/twoscomp.html
Parsing URL
Open Socket succeed
Server port set is: 4500
Connect to 127.0.0.1 successfully
Target file is /~tomf/notes/cps104/twoscomp.html
Sending request to proxy server: 127.0.0.1
Response Code: 400
File received from server successfully
File saved as: twoscomp.html
Socket closed
```

```
[william45093]@apollo3 ~/ecen602/assignment4> (13:50:20 11/18/19)
:: ./client 127.0.0.1 4500 www.codeincodeblock.com/2011/06/mini-project-snake-game-in-c.html
Parsing URL
Open Socket succeed
Server port set is: 4500
Connect to 127.0.0.1 successfully
Target file is /2011/06/mini-project-snake-game-in-c.html
Sending request to proxy server: 127.0.0.1
Response Code: 404
File received from server successfully
File saved as: mini-project-snake-game-in-c.html
Socket closed

[william45093]@apollo3 ~/ecen602/assignment4> (13:51:49 11/18/19)
:: ./client 127.0.0.1 4500 www.evanjones.ca/crc32c.html
Parsing URL
Open Socket succeed
Server port set is: 4500
Connect to 127.0.0.1 successfully
Target file is /crc32c.html
Sending request to proxy server: 127.0.0.1
Response Code: 301
File received from server successfully
File saved as: crc32c.html
Socket closed

[william45093]@apollo3 ~/ecen602/assignment4> (13:52:22 11/18/19)
:: ./client 127.0.0.1 4500 https://cesg.tamu.edu/research/
Parsing URL
Open Socket succeed
Server port set is: 4500
Connect to 127.0.0.1 successfully
Target file is /research/
Sending request to proxy server: 127.0.0.1
Response Code: 404
File received from server successfully
File saved as: research
Socket closed

[william45093]@apollo3 ~/ecen602/assignment4> (13:52:37 11/18/19)
:: ./client 127.0.0.1 4500 https://www.conference-service.com/conferences/networks.html
Parsing URL
Open Socket succeed
Server port set is: 4500
Connect to 127.0.0.1 successfully
Target file is /conferences/networks.html
Sending request to proxy server: 127.0.0.1
Response Code: 301
```

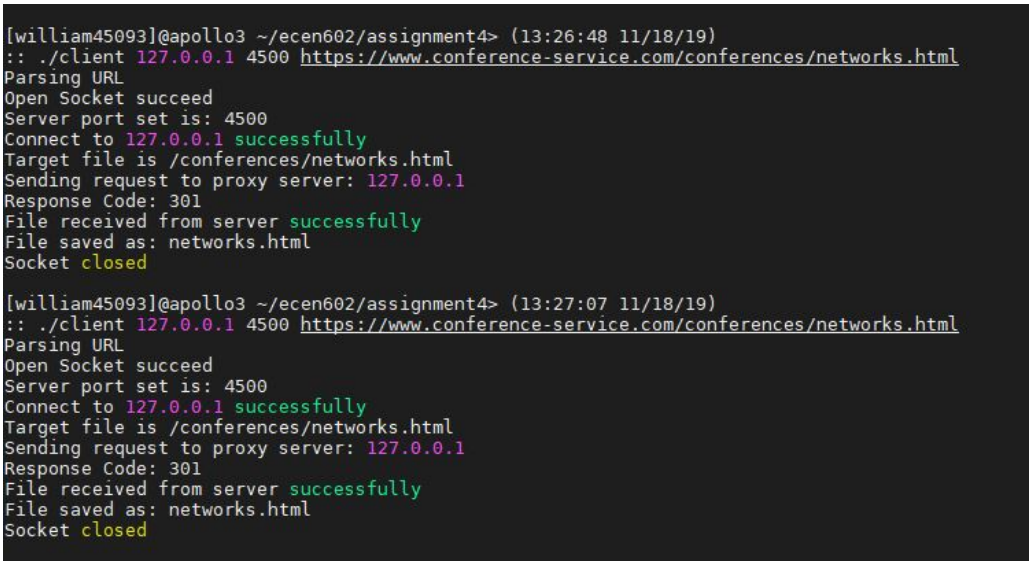
```
[william45093]@apollo3 ~/ecen602/assignment4> (13:54:41 11/18/19)
:: ./client 127.0.0.1 4500 http://ece.tamu.edu/~sunilkhatri/courses/ee449.html
Parsing URL
Open Socket succeed
Server port set is: 4500
Connect to 127.0.0.1 successfully
Target file is /~sunilkhatri/courses/ee449.html
Sending request to proxy server: 127.0.0.1
Response Code: 300
File received from server successfully
File saved as: ee449.html
Socket closed

[william45093]@apollo3 ~/ecen602/assignment4> (13:55:11 11/18/19)
:: ./client 127.0.0.1 4500 http://ece.tamu.edu/~xizhang/ECEN619/
Parsing URL
Open Socket succeed
Server port set is: 4500
Connect to 127.0.0.1 successfully
Target file is /~xizhang/ECEN619/
Sending request to proxy server: 127.0.0.1
Response Code: 200
File received from server successfully
File saved as: ECEN619
Socket closed

[william45093]@apollo3 ~/ecen602/assignment4> (13:58:55 11/18/19)
:: ./client 127.0.0.1 4500 http://www.ece.tamu.edu/~xizhang/
Parsing URL
Open Socket succeed
Server port set is: 4500
Connect to 127.0.0.1 successfully
Target file is /~xizhang/
Sending request to proxy server: 127.0.0.1
Response Code: 200
File received from server successfully
File saved as: ~xizhang
Socket closed
```

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

Server

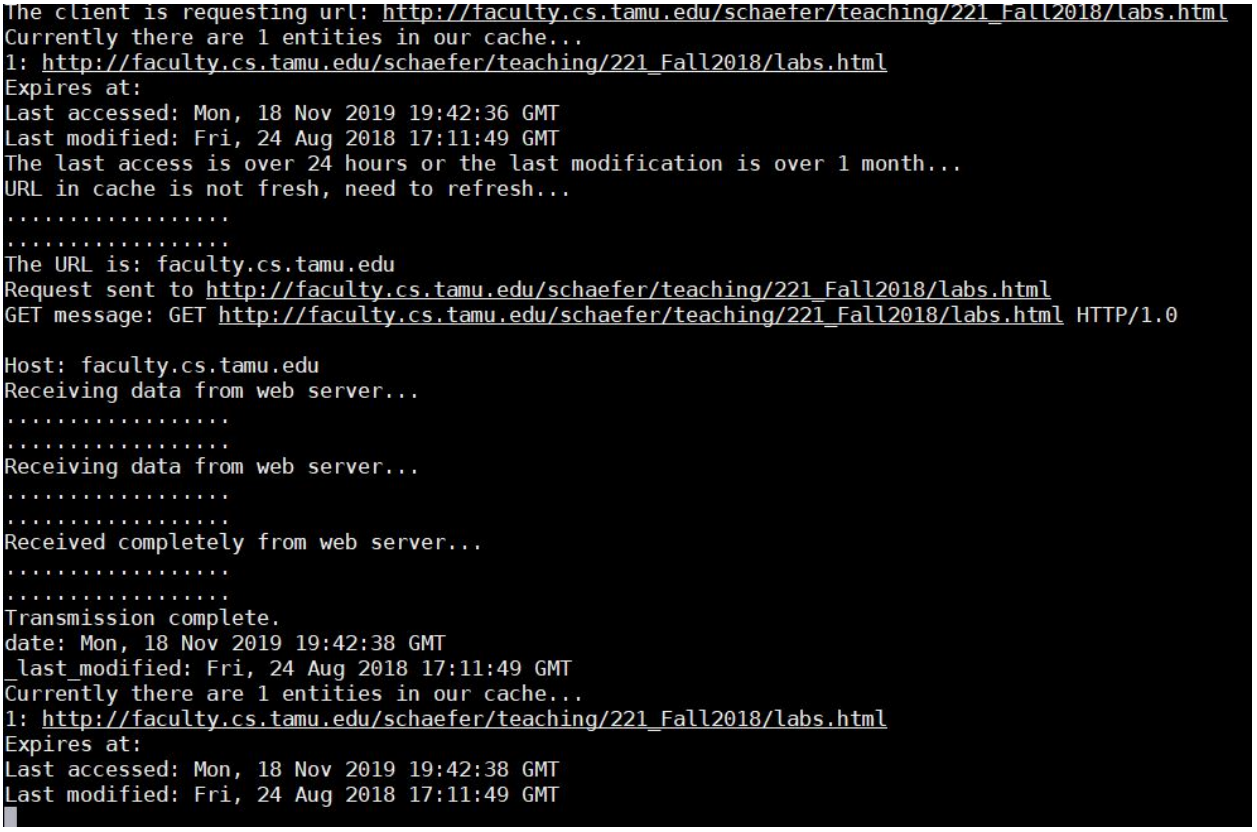



Client

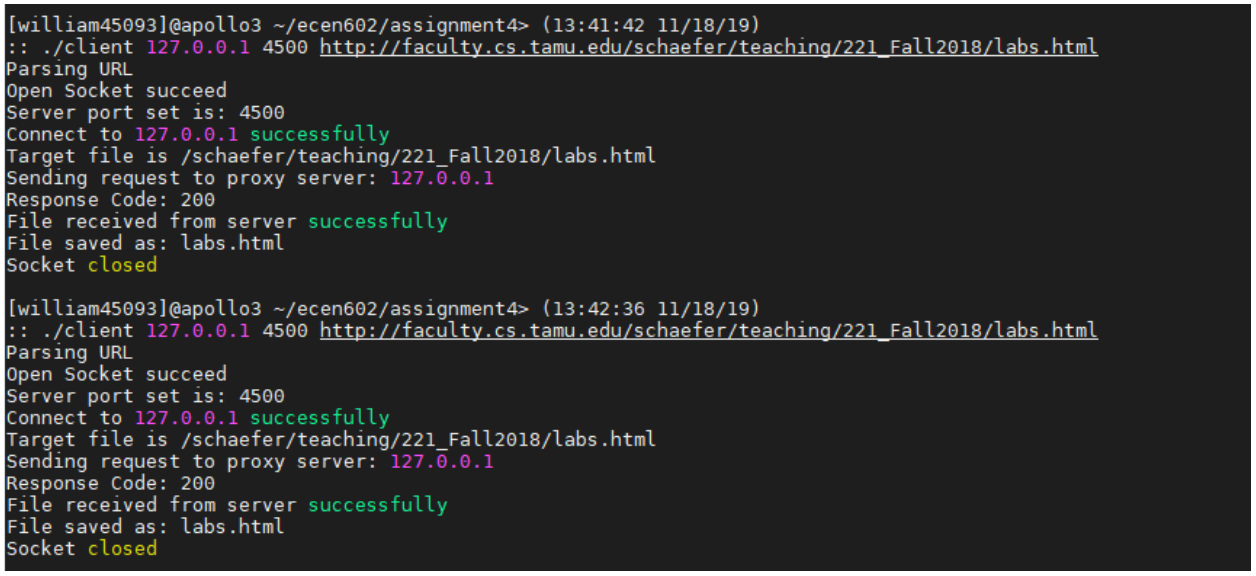


5. A stale entry in the cache without an Expires header is determined based on the last Web server access time and last modification time, the stale cache entry is replaced with fresh data, and the fresh data is delivered to the requester

Server

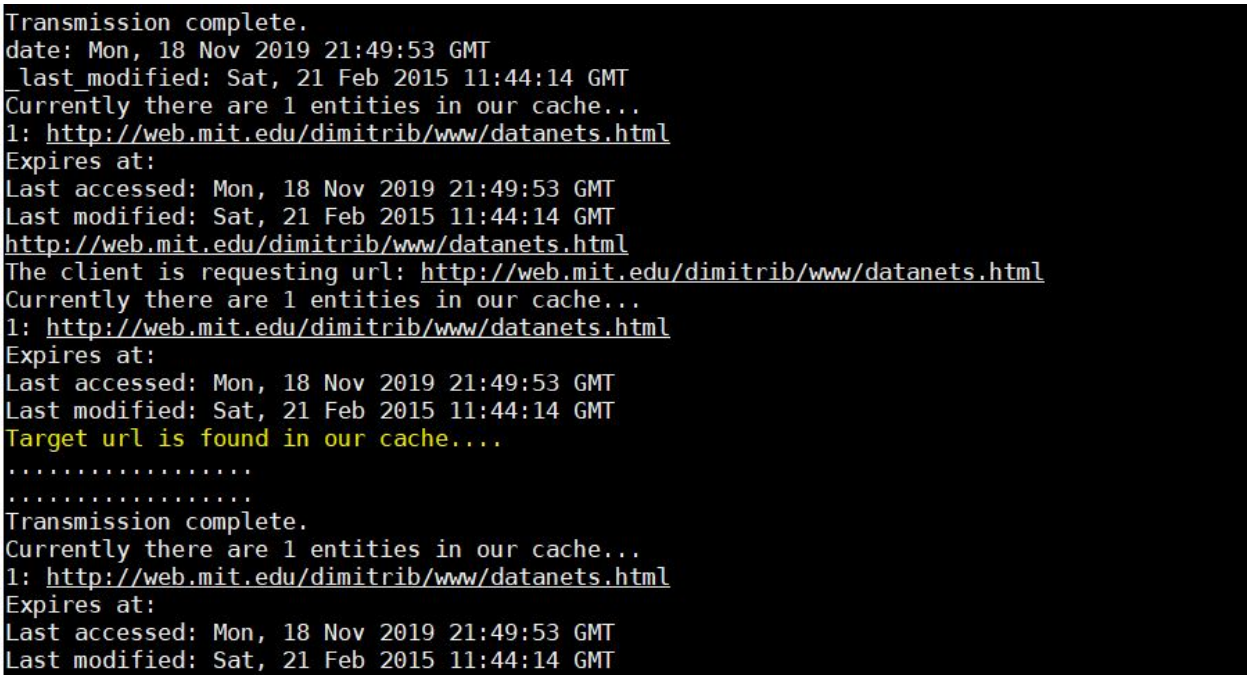



Client

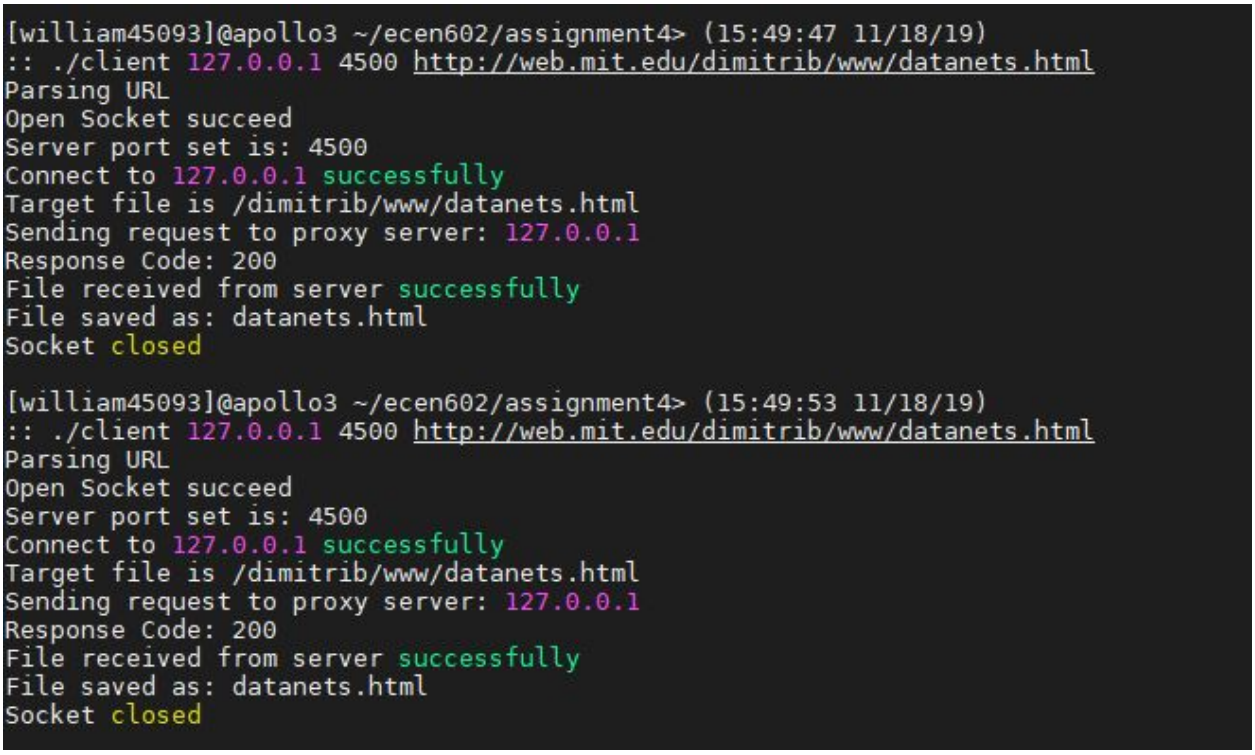


6. A cache entry without an Expires header that has been previously accessed from the Web server in the last 24 hours and was last modified more than one month ago is returned to the requester

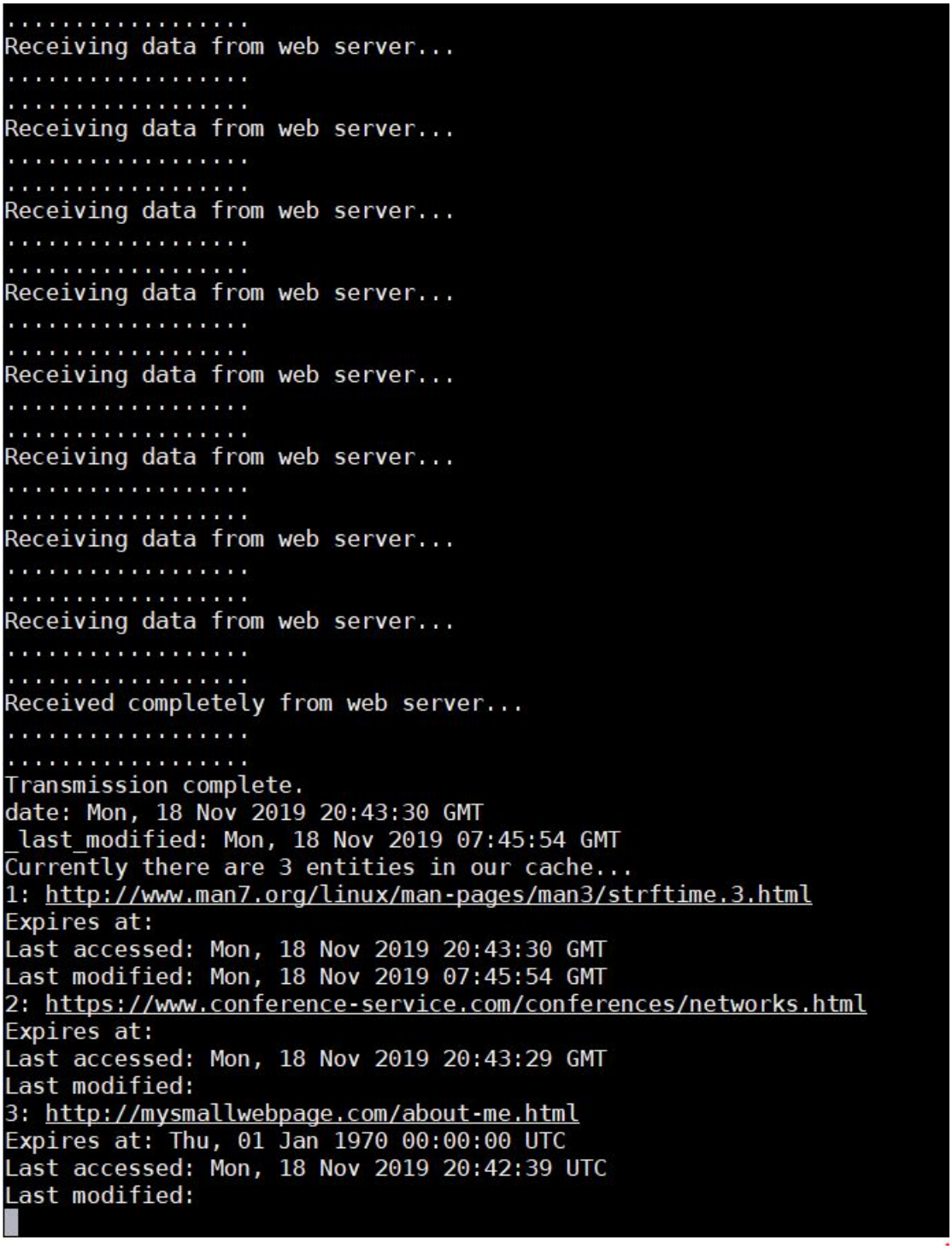
Server



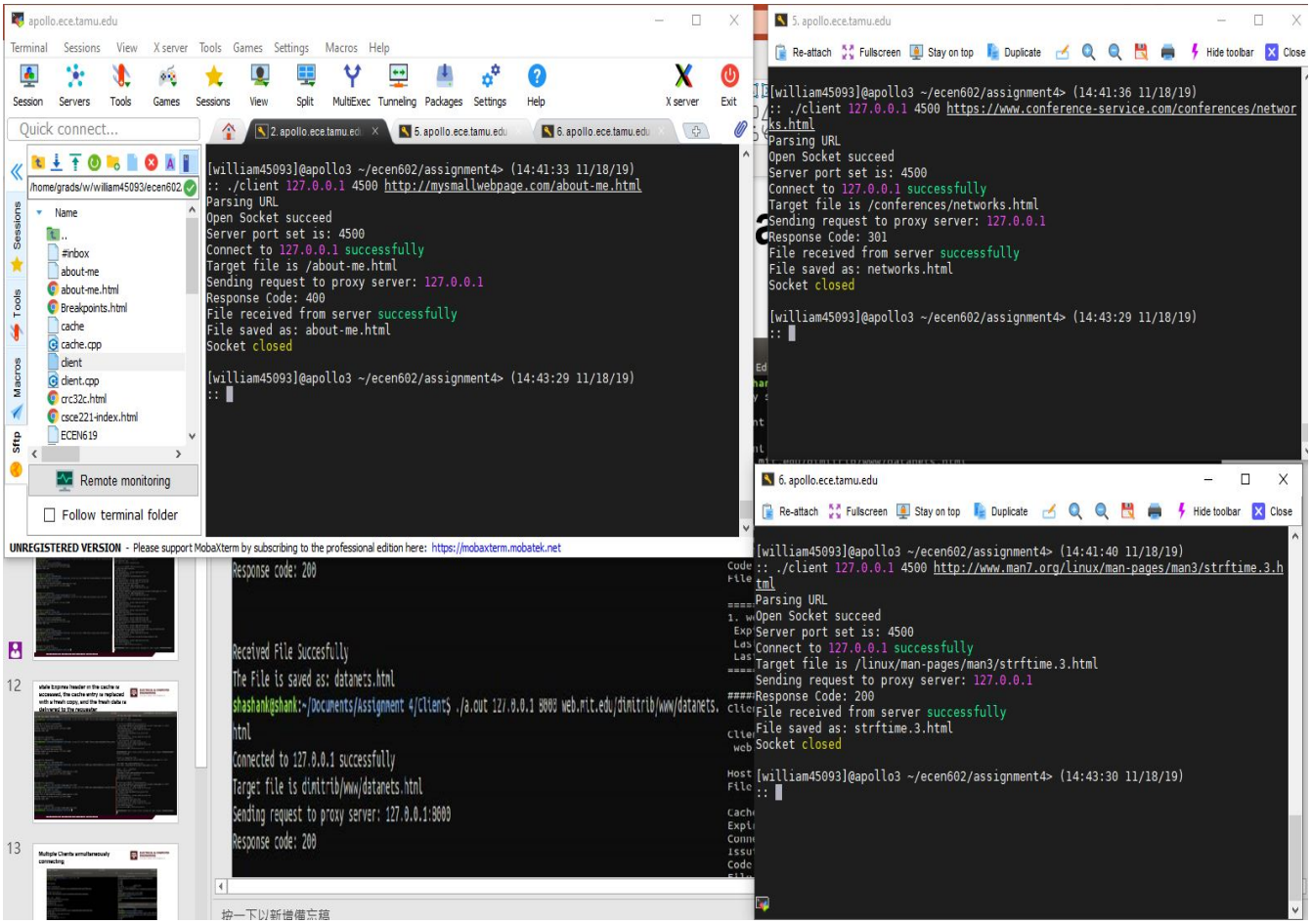

Client



7. three clients can simultaneously access the proxy server and get the correct data Server



Client




## d. Code

server.cpp

```
#include <stdlib.h>
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/time.h>
#include <netdb.h>
#include <stdlib.h>
```

```
#include <errno.h>
#include <string.h>
#include <sys/wait.h>
#include <signal.h>
#include <sys/stat.h>
#include <time.h>
using namespace std;

#define MAX_DATA_SIZE 1024
#define MAX_BUFFER_SIZE 102400

//  DESIGNED A DATA STRUCTURE CALLED NODE TO SAVE EACH ENTITY IN LRU CACHE.
struct Node{
   string key;
   string data;
   string expireat;
   string date;
   string last_modified;
   Node* next;
   Node(): key(""), data(""), expireat(""), date(""), last_modified(""), next(NULL) {}
   Node(string key, string data, string expireat, string date, string last_modified){
      this->key = key;
      this->data = data;
      this->next = NULL;
      this->expireat = expireat;
      this->date = date;
      this->last_modified = last_modified;
   }
};

//  LRU CACHE IS USED TO CACHE RECENT USED K ENTITIES. WHEN A REQUEST ARRIVES, WE
FIRSTLY CHECK IF IT IS CONTAINED IN THE CACHE. IF NOT WE SEND REQUEST TO THE WEB
SERVER
class LRUcache{
   public:
   unordered_map<string, Node*> map;
   int size;
   int capacity;
   Node* header;
   Node* tail;
   LRUcache(int capacity) {
      this->header = new Node();
      this->tail = header;
      this->size = 0;
```

```
    this->capacity = capacity;
    map.clear();
}
// MOVE THE NODE TO THE END OF END OF THE QUEUE TO MAINTAIN LRU CACHE
void movetoTail(Node* prev){
    if (prev->next == tail)
        return;
    Node* temp = prev->next;
    prev->next = temp->next;
    map[temp->next->key] = prev;
    map[temp->key] = tail;
    tail->next = temp;
    tail = tail->next;
}
Node* get(string key) {
    if (map.find(key) == map.end())
        return NULL;
    movetoTail(map[key]);
    return map[key]->next;
}

void push(string key, string data, string _expireat, string _date, string _last_modified) {
    if (map.find(key) != map.end()){
        map[key]->next->data = data;
        map[key]->next->expireat = _expireat;
        map[key]->next->date = _date;
        map[key]->next->last_modified = _last_modified;
        movetoTail(map[key]);
    }
    else{
        Node *temp = new Node(key, data, _expireat, _date, _last_modified);
        map[key] = tail;
        tail->next = temp;
        tail = tail->next;
        size += 1;
        if (size > capacity){
            Node* temp2 = header->next;
            map.erase(temp2->key);
            header->next = temp2->next;
            if (header->next != NULL)
                map[temp2->next->key] = header;
            size -= 1;
        }
    }
```

```
    }
    void print_status(){
        cout << "Currently there are " << map.size() << " entities in our cache..." << endl;
        int cnt = 0;
         for (auto &x: map){
            cout << ++cnt << ": " << x.first << endl;
            cout << "Expires at: " << map[x.first]->next->expireat << endl;
            cout << "Last accessed: " << map[x.first]->next->date << endl;
            cout << "Last modified: " << map[x.first]->next->last_modified << endl;
         }
    }
};

int findsubstr(string str, string a){
    for (int i = 7; i < str.length() - a.length(); i++){
        if (str.substr(i, a.length()) == a)
            return i;
    }
    return -1;
};

void print_status(std::string s){
    std::cout << s << "..." << std::endl;
    std::cout << "..................." << std::endl;
    std::cout << "..................." << std::endl;
}

double TimeDiffToNow (string t) {
    int t_length = t.length();
    double difference;
    char* time_in = new char[t_length + 1];
    time_t curr_time;
    time_t tm_in;
    struct tm* now;
    struct tm tm;

    // current time
    curr_time = time(NULL);
    now = gmtime(&curr_time);
    curr_time = mktime(now);
    // input time
    strcpy(time_in, t.c_str());
    strptime(time_in, "%a, %d %b %Y %H:%M:%S %Z", &tm);
    tm_in = mktime(&tm);
```

```
    difference = difftime(curr_time, tm_in);

    char* cur;
    char* in;

    delete[] time_in;
    return difference;

}

void parsing_URL(char* URL, char* host_name, char* path_name) {
    char* temp_URL = (char*) malloc(MAX_DATA_SIZE * sizeof(char));
    char* temp_host;
    char* temp_path;
    char* temp_file;
    char* temp;
    int length_host;
    int length_path;
    int length_file;

    // PARSING THE REQUESTED URL

    memset(temp_URL, 0, MAX_DATA_SIZE * sizeof(char));
    memcpy(temp_URL, URL, strlen(URL));
    if (strstr(temp_URL, "https://") != NULL) {
        temp_host = temp_URL + 8 * sizeof(char); // point to the next char after "https:\\"
    }
    else if (strstr(temp_URL, "http://") != NULL) {
        temp_host = temp_URL + 7 * sizeof(char); // point to the next char after "http:\\"
    }
    else {
        temp_host = temp_URL; // point to the head of URL if no "http:\\" included
    }
    // find the second "/" and str before it would be path name
    temp_path = strtok(temp_host, "/");
    temp_path = strtok(NULL, "/") - 1 * sizeof(char);
    memcpy(temp_URL, URL, strlen(URL));
    length_host = strlen(temp_host) - strlen(temp_path);
    length_path = strlen(temp_path) + 1;
    memcpy(host_name, temp_host, length_host);
    memcpy(path_name, temp_path, length_path);
    temp = strtok(temp_path, "/");
    // find the file name
    while (temp != NULL) {
```

```
        temp_file = temp;
        temp = strtok(NULL, "/");
    }
    free(temp_URL);
}

int main(int argc, char *argv[]){
    if (argc != 3){
        errno = EPERM;
        perror("Illegal Input! Please only input your ip address and port number. ");
        exit(EXIT_FAILURE);
    }

    string ID_addr = argv[1];
    string Port = argv[2];

    int web_socket;
    int server_socket;
    int client_socket;
    struct addrinfo currinfo, *serverinfo, *p;
    int current;
    int yes = 1;
    struct sockaddr_storage client_addr;

    fd_set master_set;
    fd_set curr_set;
    socklen_t addrlen;
    int fdmax;
    char buffer[MAX_DATA_SIZE];
    char to_get_buffer[MAX_BUFFER_SIZE];
    char to_receive_buffer[MAX_BUFFER_SIZE + 1];
    char ipv4[30];
    LRUcache mycache(10);

    char path_name[50];
    char timestamp[30];

    FD_ZERO(&master_set);
    FD_ZERO(&curr_set);
    memset(&currinfo, 0, sizeof(currinfo));
    currinfo.ai_family = AF_INET;
    currinfo.ai_socktype = SOCK_STREAM;
    currinfo.ai_flags = AI_PASSIVE;
    if (getaddrinfo(NULL, argv[2], &currinfo, &serverinfo) != 0) {
```

```
    perror("Fail to get address info");
    exit(EXIT_FAILURE);
}
for(p = serverinfo; p != NULL; p = p->ai_next){
    if ((server_socket = socket(p->ai_family, p->ai_socktype,p->ai_protocol)) < 0){
        perror("server: socket");
        continue;
    }

    if (setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int))  < 0){
        perror("Fail to create socket");
        exit(EXIT_FAILURE);
    }

    if (::bind(server_socket, p->ai_addr, p->ai_addrlen) < 0) {
        close(server_socket);
        perror("Fail to bind");
        continue;
    }
    break;
}

if (!p){
    perror("Fail to bind");
    exit(EXIT_FAILURE);
}
print_status("Socket has been created");
print_status("Socket has been binded");

if (listen(server_socket, 10) < 0)   {
    perror("Fail to listen");
    exit(EXIT_FAILURE);
}
FD_SET(server_socket, &master_set);
fdmax = server_socket;
socklen_t addr_length;
while (true){
    curr_set = master_set;
    if (select(fdmax + 1, &curr_set, NULL, NULL, NULL) < 0){
        perror("Fail to select");
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i <= fdmax; i++){
```

```
        if (FD_ISSET(i, &curr_set)){
            if (i == server_socket){
                addr_length = sizeof(client_addr);
                client_socket = accept(server_socket, (struct sockaddr *)&client_addr,
&addr_length);
                if (client_socket < 0)
                    perror("Fail to accept");
                else {
                    FD_SET(client_socket, &master_set);
                    if (client_socket > fdmax)
                        fdmax = client_socket;
                }
            }
            else{
                size_t received_size = 0 ;
                memset(buffer,0, MAX_DATA_SIZE);
                received_size = recv(i, buffer, MAX_DATA_SIZE, 0);
                for (auto &x: buffer)
                    cout << x;
                cout << endl;

                if (received_size <= 0) {
                    if (received_size == 0) {
                        print_status("No more data is received");
                    } else {
                        perror("Fail to received");
                    }
                    close(i);
                    FD_CLR(i, &master_set);
                    break;
                }

                cout << "The client is requesting url: " << buffer << endl;
                string url_in_str(buffer);
                mycache.print_status();
                Node* res = mycache.get(url_in_str);

                bool url_expired = false;
                //  IF THE URL IS IN THE CACHE, WE NEED TO CHECK IF IT EXPIRED
                if (res != NULL) {
                    if (res->expireat == "") {
                        if (((res->date) == "") || ((res->last_modified) == "")) {
                            cout << "Two of the expires, date and last_modified are missing..."
<<endl;
```

```
                    print_status("URL in cache is not fresh, need to refresh");
                    url_expired = true;
                }
                else if ((TimeDiffToNow(res->date) > 86400.00) &&
(TimeDiffToNow(res->last_modified) > 2592000.00)) {
                    cout << "The last access is over 24 hours or the last modification is over 1
month..." <<endl;
                    print_status("URL in cache is not fresh, need to refresh");
                    url_expired = true;
                }
            }
            else if (TimeDiffToNow(res->expireat) > 0.00) {
                cout << "URL in cache is expired..." <<endl;
                print_status("URL in cache is not fresh, need to refresh");
                url_expired = true;
            }
        }
        if ((res == NULL) || (url_expired == true)){
                                    if (res == NULL) {
                                        print_status("Currently the target url is not
cached");
                                    }
            memset(ipv4, 0, sizeof(ipv4));
            memset(path_name, 0, sizeof(path_name));

            parsing_URL(buffer, ipv4, path_name);
            cout<<"The URL is: " << ipv4 << endl;

            memset(&currinfo, 0, sizeof(currinfo));
            currinfo.ai_family = AF_INET;
            currinfo.ai_socktype = SOCK_STREAM;

            if (getaddrinfo(ipv4, "http", &currinfo, &serverinfo) != 0) {
                perror("Fail to get address info");
                exit(EXIT_FAILURE);
            }

            for(p = serverinfo; p != NULL; p = p->ai_next){
                if ((web_socket = socket(p->ai_family, p->ai_socktype,p->ai_protocol)) < 0){
                    perror("server: socket");
                    continue;
                }

                if (connect(web_socket, p->ai_addr, p->ai_addrlen) < 0) {
```

```
            close(web_socket);
            perror("Fail to conenct");
            continue;
        }
        break;
    }
    freeaddrinfo(serverinfo);
    memset(to_get_buffer, 0, MAX_BUFFER_SIZE);
    strcpy(to_get_buffer, "GET ");
                                    if (url_in_str.substr(0, 4) != "http")
                                            strcat(to_get_buffer, "http://");
    strcat(to_get_buffer, buffer);
    strcat(to_get_buffer, " HTTP/1.0\r\n\r\n");

    strcat(to_get_buffer, "Host: ");
    strcat(to_get_buffer, ipv4);

    received_size = send(web_socket, to_get_buffer, sizeof(to_get_buffer), 0);
    cout << "Request sent to " << buffer << endl;
    cout << "GET message: " << to_get_buffer << endl;
    memset(to_receive_buffer, 0, MAX_BUFFER_SIZE);
    received_size = 0;
    bool received = true;
    char *read_ptr;
    read_ptr = to_receive_buffer;
    size_t sent_size;
    while (received){
        received_size = recv(web_socket, read_ptr, MAX_DATA_SIZE*sizeof(char),
0);
        if (strstr(read_ptr, "404") != NULL){
            print_status("404 Not Found");
            sent_size = send(web_socket, "404 Not Found", 10, 0);
            close(web_socket);
            received = false;
            break;
        }
        if (received_size <= 0){
            print_status("Received completely from web server");
            close(web_socket);
            received = false;
            break;
        }
        print_status("Receiving data from web server");
        read_ptr += received_size;
```

```
}
sent_size = send(i, to_receive_buffer, sizeof(to_receive_buffer), 0);
if (sent_size <= 0){
   perror("Fail to send");
   exit(EXIT_FAILURE);
}
cout << "Transmission complete." << endl;

// PARSING FOR EXPIRES, DATE, LAST_MODIFIED
char* ptr;
string _expireat, _date, _last_modified;
if ((ptr = strstr(to_receive_buffer, "expires:")) != NULL) {
   memset(timestamp, 0, sizeof(timestamp));
   memcpy(timestamp, ptr + 9, sizeof(timestamp));
   _expireat = timestamp;
   cout << "expires: " << _expireat << endl;
}
else if ((ptr = strstr(to_receive_buffer, "Expires:")) != NULL) {
   memset(timestamp, 0, sizeof(timestamp));
   memcpy(timestamp, ptr + 9, sizeof(timestamp));
   _expireat = timestamp;
   cout << "expires: " << _expireat << endl;
}
else {
   _expireat = "";
}

if ((ptr = strstr(to_receive_buffer, "date:")) != NULL) {
   memset(timestamp, 0, sizeof(timestamp));
   memcpy(timestamp, ptr + 6, sizeof(timestamp));
   _date = timestamp;
   cout << "date: " << _date << endl;
}
else if ((ptr = strstr(to_receive_buffer, "Date:")) != NULL) {
   memset(timestamp, 0, sizeof(timestamp));
   memcpy(timestamp, ptr + 6, sizeof(timestamp));
   _date = timestamp;
   cout << "date: " << _date << endl;
}
else {
   _date = "";
}

if ((ptr = strstr(to_receive_buffer, "Last-Modified:")) != NULL) {
```

```
                    memset(timestamp, 0, sizeof(timestamp));
                    memcpy(timestamp, ptr + 15, sizeof(timestamp));
                    _last_modified = timestamp;
                    cout << "_last_modified: " << _last_modified << endl;
                }
                else {
                    _last_modified = "";
                }
                // IF EXPIRES AND LAST-MODIFIED ARE MISSING, NOT CACHE THE URL (WE
ARE UNSURE IF WE NEED TO CACHE IT IN THIS CASE)
                // if ((_expireat == "") &&(_last_modified =="")) {
                //     print_status("Expires and Last-Modified are missing, the URL will not be
cached");
                //     close(i);
                //     FD_CLR(i, &master_set);
                //     break;
                // }
                mycache.push(url_in_str, to_receive_buffer, _expireat, _date,
_last_modified);
                mycache.print_status();
                memset(buffer, 0, MAX_DATA_SIZE);
                memset(to_get_buffer, 0, MAX_BUFFER_SIZE);
                close(i);
                FD_CLR(i, &master_set);
                break;
            }
            else{
                print_status("Target url is found in our cache.");
                strcpy(to_receive_buffer, res->data.c_str());
                int sent = send(i, to_receive_buffer, sizeof(to_receive_buffer), 0);
                if (sent <= 0){
                    perror("Fail to send");
                    exit(EXIT_FAILURE);
                }
                cout << "Transmission complete." << endl;
                mycache.print_status();
                memset(buffer, 0, MAX_DATA_SIZE);
                memset(to_get_buffer, 0, MAX_BUFFER_SIZE);
                close(i);
                FD_CLR(i, &master_set);
                break;
            }
        }
    }
```

```
      }
   }
   return 0;
}
```

client.cpp

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <iostream>

#define MAX_BUFFER_SIZE 1024 * 1024

using namespace std;

void parsing_URL(char* URL, char* host_name, char* path_name, char* file_name) {
    char* temp_URL = (char*) malloc(150 * sizeof(char));
    char* temp_host;
    char* temp_path;
    char* temp_file;
    char* temp;
    int length_host;
    int length_path;
    int length_file;

    // Parsing the requested URL

    memset(temp_URL, 0, 150 * sizeof(char));
    memcpy(temp_URL, URL, strlen(URL));
    if (strstr(temp_URL, "https://") != NULL) {
            temp_host = temp_URL + 8 * sizeof(char); // point to the next char after
"https:\\"
    }
    else if (strstr(temp_URL, "http://") != NULL) {
            temp_host = temp_URL + 7 * sizeof(char); // point to the next char after
"http:\\"
```

```
    }
    else {
            temp_host = temp_URL; // point to the head of URL if no "http:\\" included
    }
    // find the second "/" and str before it would be path name
    temp_path = strtok(temp_host, "/");
    temp_path = strtok(NULL, "/") - 1 * sizeof(char);
    memcpy(temp_URL, URL, strlen(URL));
    length_host = strlen(temp_host) - strlen(temp_path);
    length_path = strlen(temp_path) + 1;
    memcpy(host_name, temp_host, length_host);
    memcpy(path_name, temp_path, length_path);
    temp = strtok(temp_path, "/");
    // find the file name
    while (temp != NULL) {
            temp_file = temp;
            temp = strtok(NULL, "/");
    }
    temp_file = temp_file;
    memcpy(temp_URL, URL, strlen(URL));
    length_file = strlen(temp_file);
    memcpy(file_name, temp_file, length_file);

    free(temp_URL);

}



int main(int argc, char **argv){

    if (argc != 4) {
            errno = EPERM;
            printf("INPUT_ERROR: ERRNO: \t%s\n", strerror(errno));
            return -1;
    }
    char* server_name = argv[1];
    char* _server_port = argv[2];
    char* _URL_name = argv[3];


    int server_port = -1;
    int c;
   int socketfd = -1;
```

```
  char host_name[50];
char path_name[100];
  char file_name[50];
  char GET_msg[150];

struct sockaddr_in server_addr;

  char buffer[MAX_BUFFER_SIZE + 1];
  int length_recv = 0;
  char response_code[3];
  char* file_head;
  int file_length;
  char* head_response;

  // Parsing the requested URL
  printf("Parsing URL\n");
  parsing_URL(_URL_name, host_name, path_name, file_name);

  if (file_name[strlen(file_name) - 1] == '/') {
          file_name[strlen(file_name) - 1] = '\0';
  }

if ((socketfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
   printf("SOCKET_OPEN: ERRNO: \t%s\n", strerror(errno));
          return -1;
}
  printf("Open Socket succeed\n");

// Input the server port to be connected
  server_port = atoi(_server_port);
  printf("Server port set is: %d\n", server_port);

// Setup the server address

memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(server_port);
if (inet_pton(AF_INET, server_name, &server_addr.sin_addr) < 0) {
   printf("ADDR_TRANS: ERRNO: \t%s\n", strerror(errno));
          close(socketfd);
          return -1;
}

// Connect the socket to the server
```

```
if ((connect(socketfd, (struct sockaddr *)&server_addr, sizeof(server_addr))) < 0) {
   printf("CONNECT: RRNO: \t%s\n", strerror(errno));
            close(socketfd);
            return -1;
}
  printf("Connect to %s successfully\n", server_name);

  // Create GET message
  sprintf(GET_msg, "GET %s HTTP/1.0\r\nHOST: %s\r\n\r\n", path_name, host_name);

  // Send GET request to proxy server
  if ((write(socketfd, _URL_name, strlen(_URL_name))) < 0) {
   printf("Send GET message: ERRNO: \t%s\n", strerror(errno));
            close(socketfd);
   return -1;
}
  //printf("******GET message was sent******\n%s", GET_msg);
  printf("Target file is %s\n", path_name);
  printf("Sending request to proxy server: %s\n", server_name);

  // Received requested file from socket
  memset(buffer, 0, (MAX_BUFFER_SIZE * sizeof(char) + 1));

  // Received file and save in the buffer
  length_recv = recv(socketfd, buffer, MAX_BUFFER_SIZE * sizeof(char), 0);

  if (length_recv < 0) {
            printf("Recv: ERRNO: \t%s\n", strerror(errno));
            close(socketfd);
            return -1;
  }
  // If no data received, close socket and return
  else if (length_recv == 0) {
            printf("No data received from server\n");
            close(socketfd);
            return 0;
  }
  // Show response code
  if ((head_response = strstr(buffer, "HTTP/1.0")) != NULL){
            memcpy(response_code, head_response + 9, 3 * sizeof(char));
  }
  else if ((head_response = strstr(buffer, "HTTP/1.1")) != NULL){
            memcpy(response_code, head_response + 9, 3 * sizeof(char));
  }
```

```
    printf("Response Code: %s\n", response_code);

    printf("File received from server successfully\n");
    // Create file
    FILE* fd = fopen(file_name, "w");

    file_head = buffer;
    file_length = strlen(buffer);

    // Write the buffer to the file
    fwrite(file_head, sizeof(char), file_length, fd);
    printf("File saved as: %s\n", file_name);

    fclose(fd);
    close(socketfd);
    printf("Socket closed\n");

    return 0;
}
```