ECEN 602

NETWORK SIMULATION ASSIGNMENT – 04

TEAM 17

Mohammad Faisal Khan

Amiya Ranjan Panda

****
****
****

README

****
****
****

TITLE : Implementation of a simple HTTP proxy server and HTTP command line client based on RFC 1945 with additional caching feature.

INTRODUCTION :

This code is a part of the Network simulation Assignment for ECEN 602 at Texas A&M University.

It has been sucessfully compiled, executed and tested on gcc compiler (part of standard LINUX).

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems.

HTTP functions as a request–response protocol in the client–server computing model.

A web browser, for example, may be the client and an application running on a computer hosting a website may be the server. The client

submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content, or performs

other functions on behalf of the client, returns a response message to the client. The response contains completion status information

about the request and may also contain requested content in its message body. HTTP resources are identified and located on the network

by Uniform Resource Locators (URLs), using the Uniform Resource Identifiers (URI's) schemes http and https. URIs and hyperlinks in HTML

documents form interlinked hypertext documents.

HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a server

(typically port 80, occasionally port 8080; see List of TCP and UDP port numbers). An HTTP server listening on that port waits for a

client's request message. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message

of its own. The body of this message is typically the requested resource, although an error message or other information may also be

returned. In case of the web proxy server, it behaves like a server in the client-proxy interface and as a client in the

proxy-server interface.

In this implementation, When the proxy server receives a client request, it will first check its cached data in an attempt to serve

the request. If there is not a valid cache entry, however, (1) the request will be proxied to the intended destination, (2) the response

will be sent by the proxy to the client, and (3) the response will also be cached by the proxy for later use. Your proxy cache should

maintain at least 10 document entries in the cache. Entries should be replaced in a Least Recently Used (LRU) fashion.

Common Errors and Catches:

-If data is not input correctly on the command line as per the ordering given below, it throws a segmentation error.

-This is NOT to be assumed as an error.

-If data is missing from the command line, it throws segmentation error too.

-This is an iterative server, fork() is not used instead select() is used.

-The well-known socket for the HTTP server is port number 80 but the server uses a different one(asked in argument) and the connection is established in ephemeral port

as negotiated by the web server. This is done to avoid the creation of invariance in the proxy server processing.

-The url format is "www.abcd.com/path".

Usage:

1. 'make clean' to remove all previously created object files.

2. 'make' to compile the Server source code.

3. ./proxy <ip to bind> <port to bind> to run the proxy server first.

4. ./client <proxy address> <proxy port> <url to retrieve> to run the client.

steps :

1. make

2. ./proxy <ip to bind> <port to bind>

3. ./client <proxy address> <proxy port> <url to retrieve>

Package content:

1. proxy.c

2. client.c

3. Makefile

Tests:

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

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

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.

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.

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

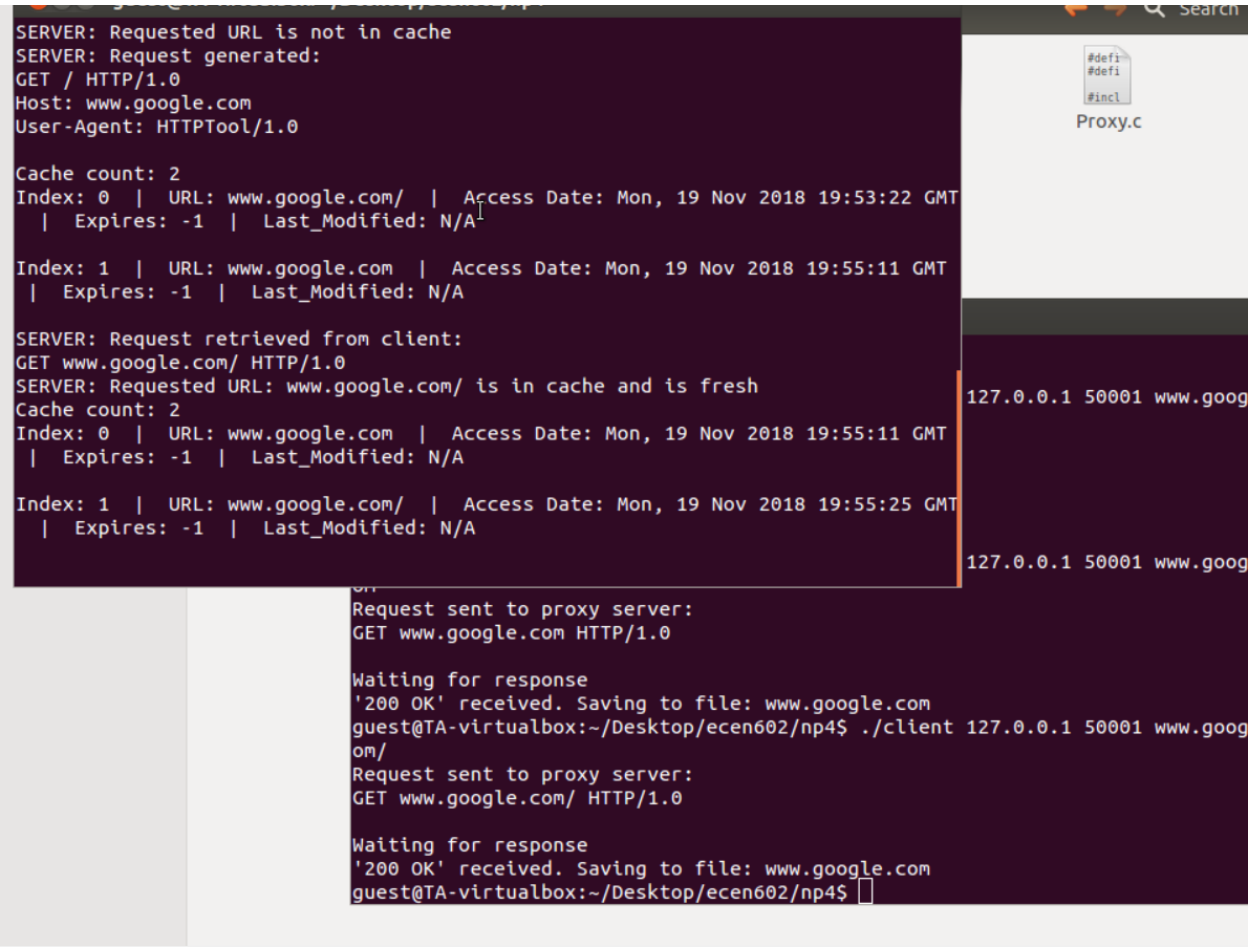
cache entry is replaced with fresh data, and the fresh data is delivered to the requester.

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

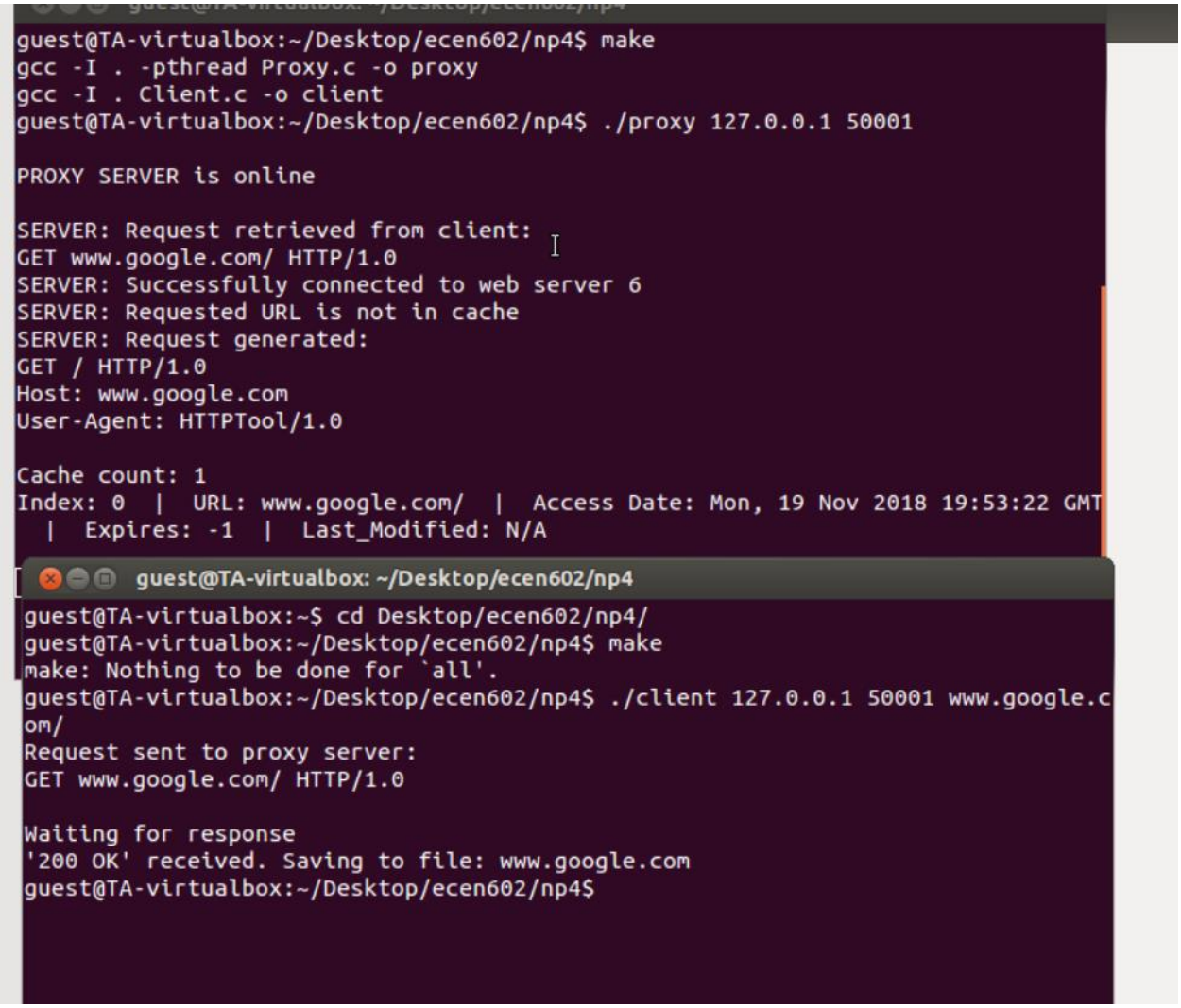
one month ago is returned to the requester.

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

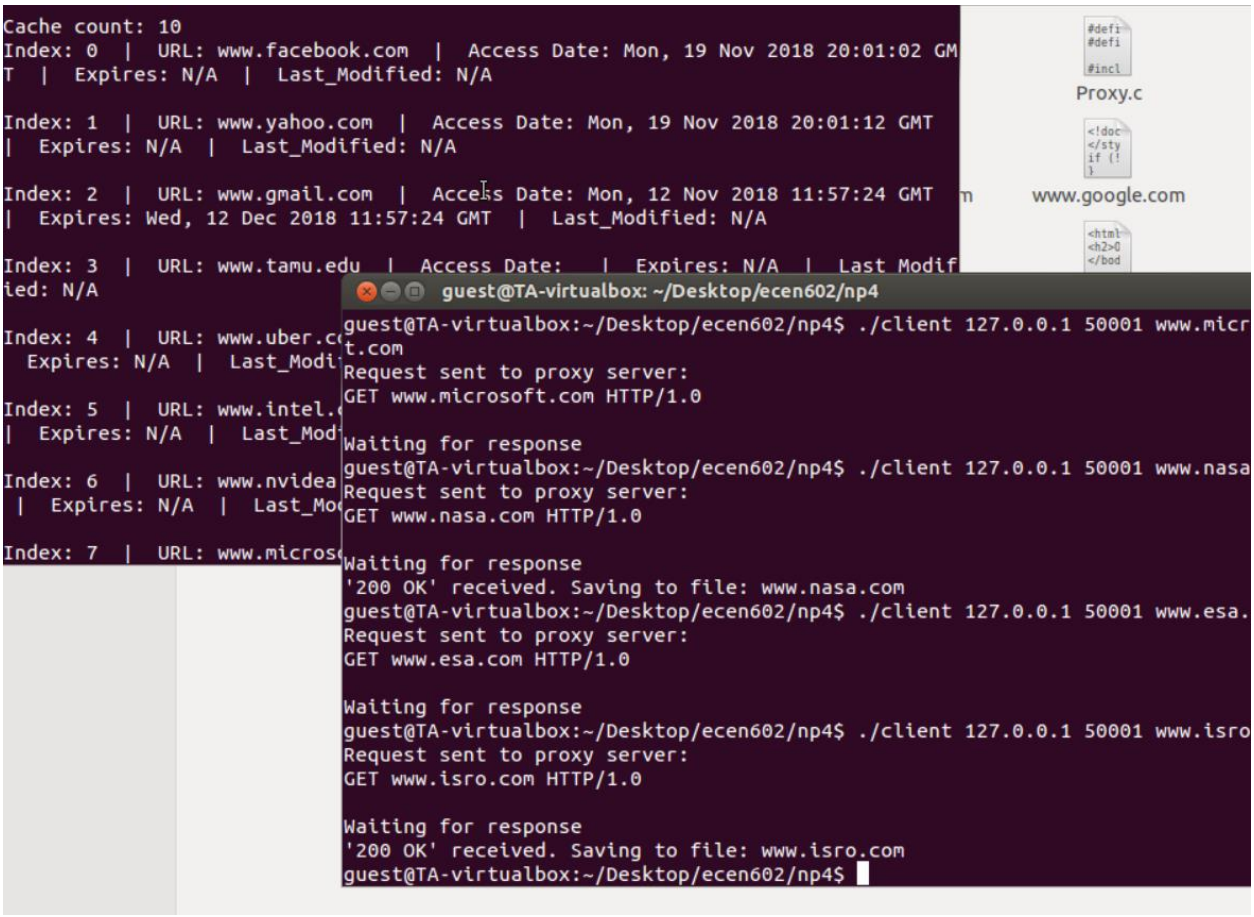
A cache hit returns the saved data to the requester.

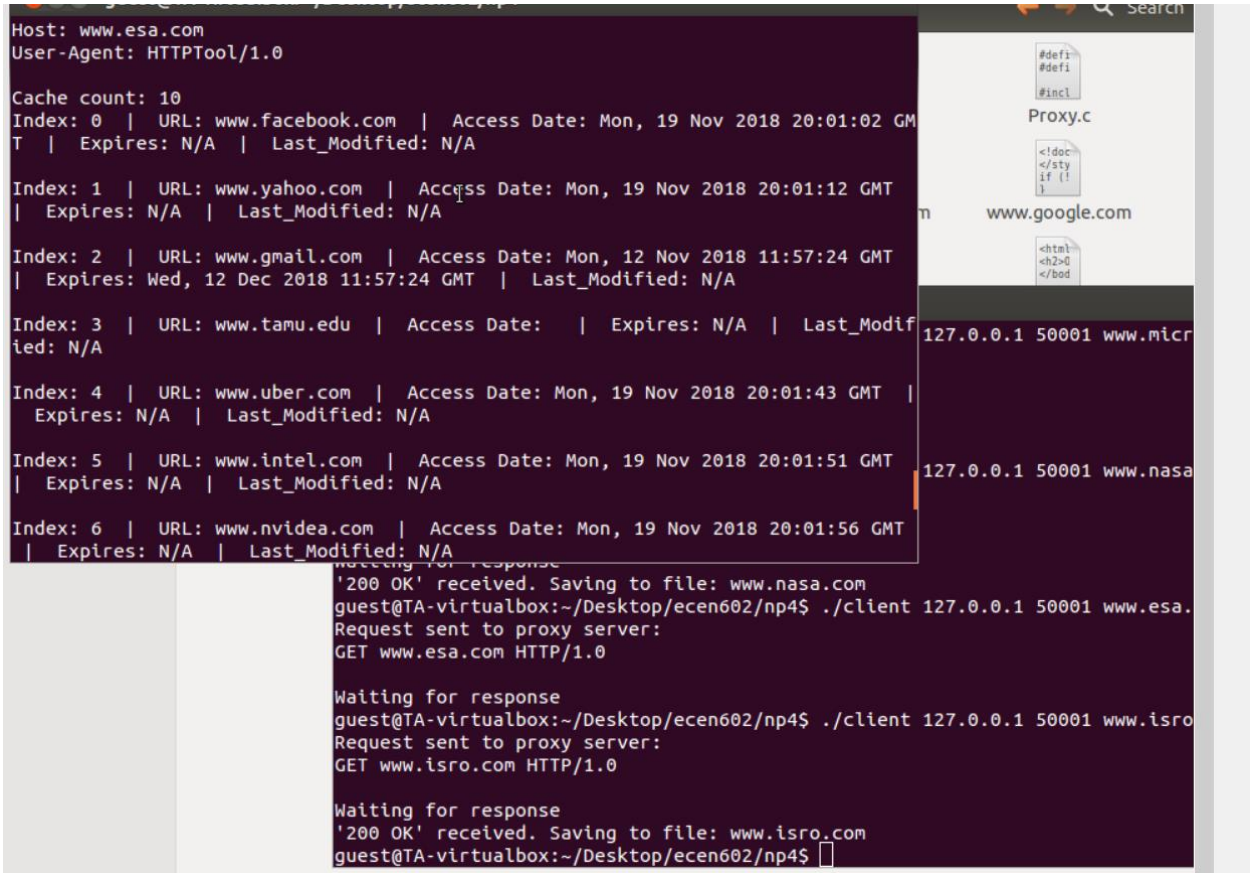


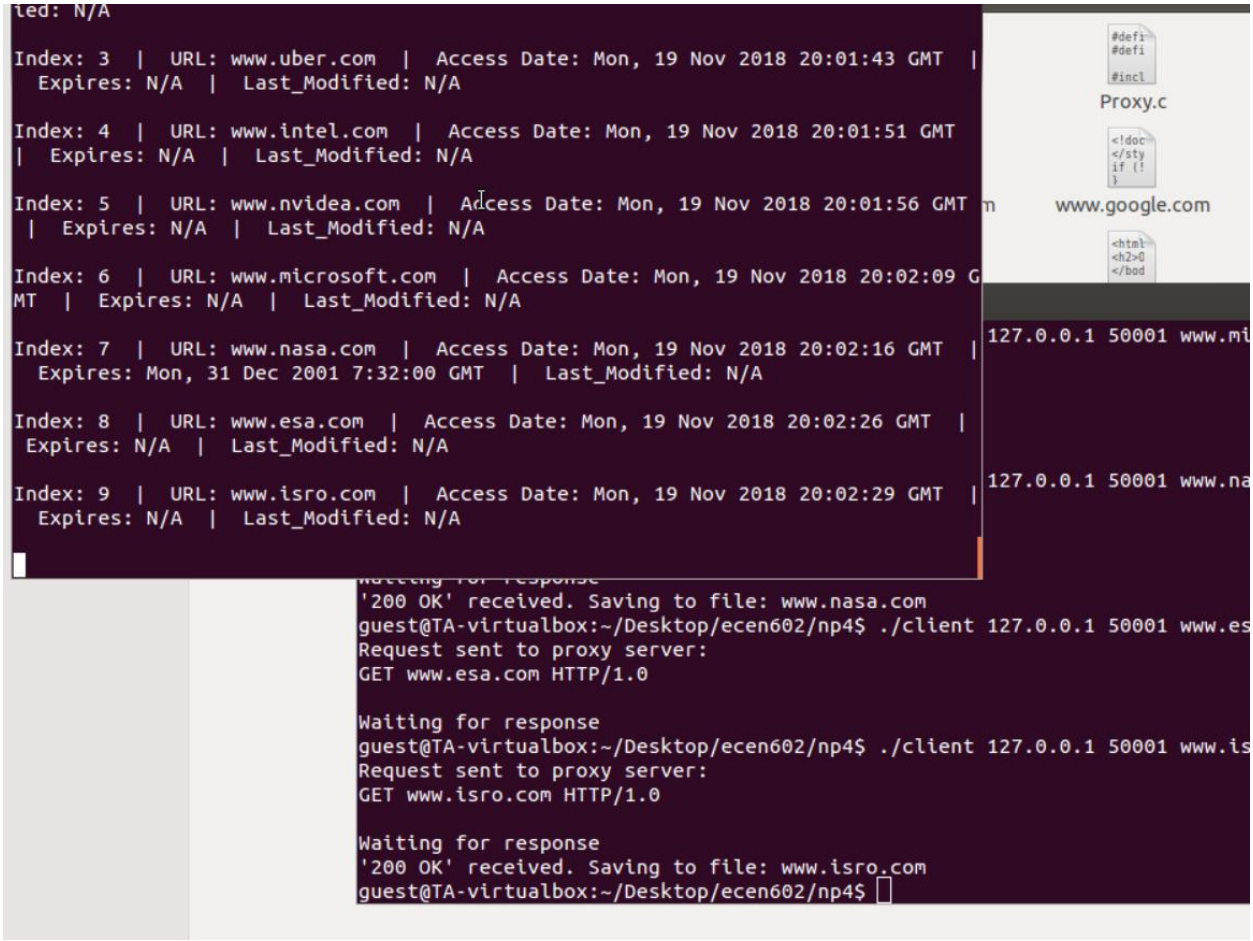
A request that is not in the cache is proxied, saved in the cache, and returned to the requester.



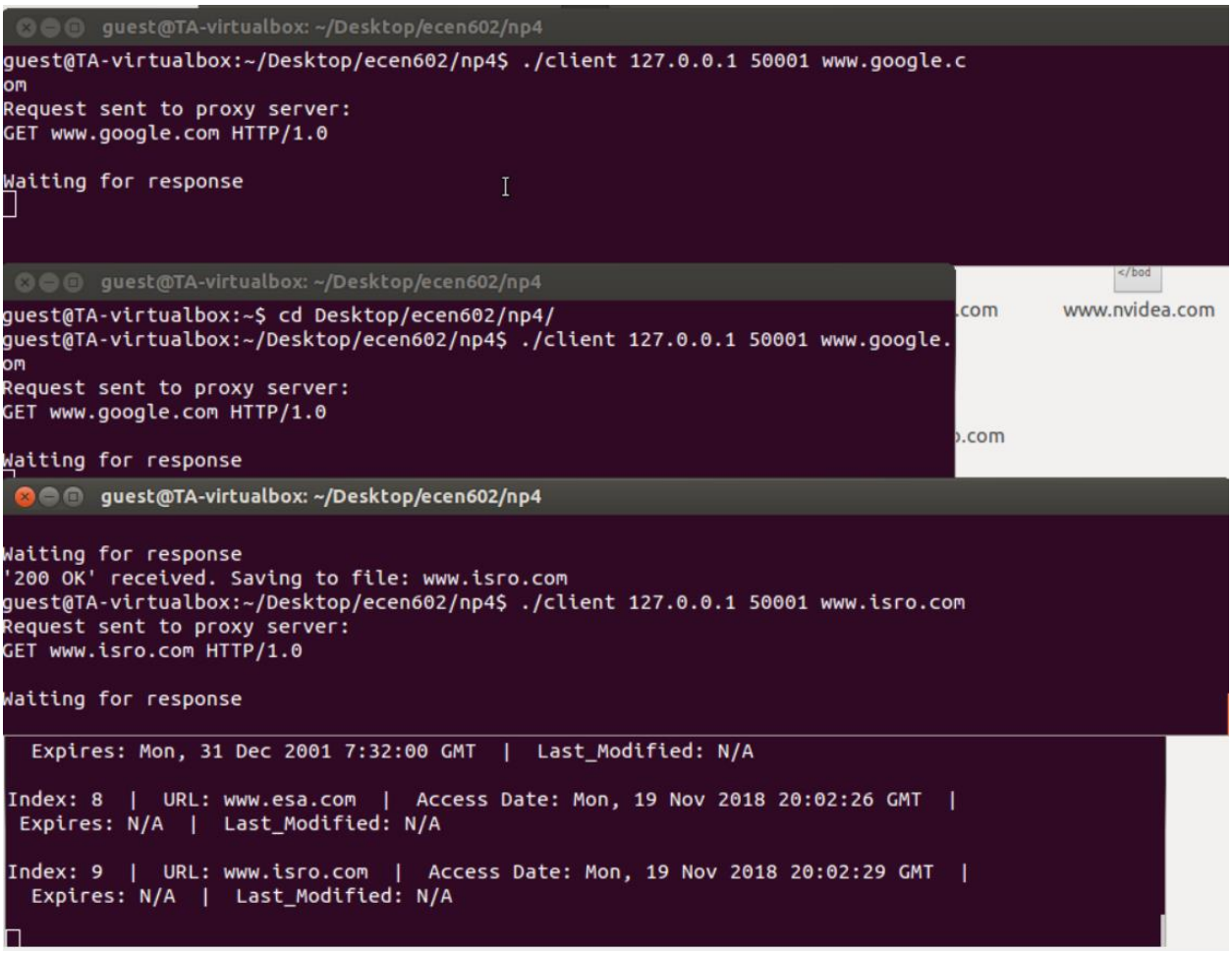
A cache miss with 10 items already in the cache is proxied, saved in the LRU location in cache, and the data is returned to the requester.




Host: www.esa.com
User-Agent: HTTPTool/1.0
Cache count: 10
Index: 0 | URL: www.facebook.com | Access Date: Mon, 19 Nov 2018 20:01:02 GMT | Expires: N/A | Last_Modified: N/A
Index: 1 | URL: www.yahoo.com | Access Date: Mon, 19 Nov 2018 20:01:12 GMT | Expires: N/A | Last_Modified: N/A
Index: 2 | URL: www.gmail.com | Access Date: Mon, 12 Nov 2018 11:57:24 GMT | Expires: Wed, 12 Dec 2018 11:57:24 GMT | Last_Modified: N/A
Index: 3 | URL: www.tamu.edu | Access Date: | Expires: N/A | Last_Modified: N/A
Index: 4 | URL: www.uber.com | Access Date: Mon, 19 Nov 2018 20:01:43 GMT | Expires: N/A | Last_Modified: N/A
Index: 5 | URL: www.intel.com | Access Date: Mon, 19 Nov 2018 20:01:51 GMT | Expires: N/A | Last_Modified: N/A
Index: 6 | URL: www.nvidea.com | Access Date: Mon, 19 Nov 2018 20:01:56 GMT | Expires: N/A | Last_Modified: N/A
Proxy.c
www.google.com
127.0.0.1 50001 www.micr
127.0.0.1 50001 www.nasa
'200 OK' received. Saving to file: www.nasa.com
guest@TA-virtualbox:~/Desktop/ecen602/np4$ ./client 127.0.0.1 50001 www.esa.
Request sent to proxy server:
GET www.esa.com HTTP/1.0
Waiting for response
guest@TA-virtualbox:~/Desktop/ecen602/np4$ ./client 127.0.0.1 50001 www.isro
Request sent to proxy server:
GET www.isro.com HTTP/1.0
Waiting for response
'200 OK' received. Saving to file: www.isro.com
guest@TA-virtualbox:~/Desktop/ecen602/np4$


ted: N/A
Index: 3 | URL: www.uber.com | Access Date: Mon, 19 Nov 2018 20:01:43 GMT |
Expires: N/A | Last_Modified: N/A
Index: 4 | URL: www.intel.com | Access Date: Mon, 19 Nov 2018 20:01:51 GMT
| Expires: N/A | Last_Modified: N/A
Index: 5 | URL: www.nvidea.com | Access Date: Mon, 19 Nov 2018 20:01:56 GMT
| Expires: N/A | Last_Modified: N/A
Index: 6 | URL: www.microsoft.com | Access Date: Mon, 19 Nov 2018 20:02:09 G
MT | Expires: N/A | Last_Modified: N/A
Index: 7 | URL: www.nasa.com | Access Date: Mon, 19 Nov 2018 20:02:16 GMT |
Expires: Mon, 31 Dec 2001 7:32:00 GMT | Last_Modified: N/A
Index: 8 | URL: www.esa.com | Access Date: Mon, 19 Nov 2018 20:02:26 GMT |
Expires: N/A | Last_Modified: N/A
Index: 9 | URL: www.isro.com | Access Date: Mon, 19 Nov 2018 20:02:29 GMT |
Expires: N/A | Last_Modified: N/A
Proxy.c
www.google.com
127.0.0.1 50001 www.mi
127.0.0.1 50001 www.na
'200 OK' received. Saving to file: www.nasa.com
guest@TA-virtualbox:~/Desktop/ecen602/np4$ ./client 127.0.0.1 50001 www.es
Request sent to proxy server:
GET www.esa.com HTTP/1.0
Waiting for response
guest@TA-virtualbox:~/Desktop/ecen602/np4$ ./client 127.0.0.1 50001 www.is
Request sent to proxy server:
GET www.isro.com HTTP/1.0
Waiting for response
'200 OK' received. Saving to file: www.isro.com
guest@TA-virtualbox:~/Desktop/ecen602/np4$

Three clients can simultaneously access the proxy server and get the correct data



CODE :

PROXY :

```
#define __USE_XOPEN 1
#define _XOPEN_SOURCE 700

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <strings.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>
#include <dirent.h>
#include <pthread.h>
#include <time.h>

#define MAX_CACHE_ENTRY 10
#define MAX_LEN 1024

struct Cache {
  char URL[256];
  char Last_Modified[50];
  char Access_Date[50];
  char Expires[50];
  char *body;
};

static const struct Cache Clear_Entry;
int num_cache_entries = 0;


struct Cache Proxy_Cache[MAX_CACHE_ENTRY];

int parse_URL (char* URL, char *hostname, int *port, char *path) {
  char *token;
  char *host_temp, *path_temp;
  char *tmp1, *tmp2;
  int num = 0;
  char s[16];
  if (strstr(URL,"http") != NULL){
    token = strtok(URL, ":");
```

```
    tmp1 = token + 7;
  }
  else{
    tmp1 = URL;
  }
  tmp2 = malloc (64);
  memcpy(tmp2, tmp1, 64);
  if(strstr(tmp1, ":") != NULL){
    host_temp = strtok(tmp1, ":");
    *port = atoi(tmp1 + strlen(host_temp) + 1);
    sprintf(s, "%d", *port);
    path_temp = tmp1 + strlen(host_temp) + strlen(s) + 1;
  }
  else{
    host_temp = strtok(tmp1, "/");
    *port = 80;
    path_temp = tmp2 + strlen(host_temp);
  }
  if (strcmp(path_temp, "") == 0)
    strcpy(path_temp, "/");
  memcpy(hostname, host_temp, 64);
  memcpy(path, path_temp, 256);
  return(0);
}

int parseHDR(const char* hdr, char* buf, char* op) {
  char *st = strstr(buf, hdr);
  if(!st) {
    return 0;
  }
  char *end = strstr(st, "\r\n");
  st += strlen(hdr);
  while(*st == ' ')
    ++st;
  while(*(end - 1) == ' ')
    --end;
  strncpy(op, st, end - st);
  op[end - st] = '\0';
  return 1;

}

int err_sys(const char* x)    // Error display source code
{
  perror(x);
  exit(1);
}

int Extract_Read(int fd, char *msg) {            // Extracts message
body from read socket

  int total = 0;
```

```
  char buffer[MAX_LEN] = {0};
  int cnt = 1;
  int h;
  while(cnt>0) {
    memset(buffer, 0, sizeof(buffer));
    cnt = read(fd, buffer, MAX_LEN);
    if (cnt == 0) break;
    strcat(msg, buffer);
    total = total + cnt;
    if (buffer[cnt - 1] == EOF) {
      strncpy(msg,msg,(strlen(msg)-1));
      total--;
      break;
    }
  }
  return total;
}

int Update_Cache(char *URL, char *buf, int flag, int x) {

  int j=0;
  int p=0;

  if (flag == 1) {                    // New entry
    if (num_cache_entries==MAX_CACHE_ENTRY){
      Proxy_Cache[0] = Clear_Entry;     // Popping LRU
      for (j=0; j<MAX_CACHE_ENTRY; j++){
        if (j+1!=MAX_CACHE_ENTRY)
          Proxy_Cache[j] = Proxy_Cache[j+1];
        else {
          // Add new entry at the head (latest)
          memset(&Proxy_Cache[j], 0, sizeof(struct Cache));
          memcpy(Proxy_Cache[j].URL,URL,256);
       Proxy_Cache[j].body = (char *) malloc(strlen(buf));
          memcpy(Proxy_Cache[j].body,buf,strlen(buf));
          parseHDR("Expires:", buf, Proxy_Cache[j].Expires);
       parseHDR("Last-Modified:", buf, Proxy_Cache[j].Last_Modified);
       parseHDR("Date:", buf, Proxy_Cache[j].Access_Date);
        }
      }
    }
    else {              // If cache has not reached max allowed
capacity (MAX_CACHE_ENTRY)
      Proxy_Cache[num_cache_entries] = Clear_Entry;
      memcpy(Proxy_Cache[num_cache_entries].URL,URL,256);
      parseHDR("Expires:", buf,
Proxy_Cache[num_cache_entries].Expires);
      parseHDR("Last-Modified:", buf,
Proxy_Cache[num_cache_entries].Last_Modified);
      parseHDR("Date:", buf,
Proxy_Cache[num_cache_entries].Access_Date);
```

```
      Proxy_Cache[num_cache_entries].body = (char *)
malloc(strlen(buf));
      memcpy(Proxy_Cache[num_cache_entries].body,buf,strlen(buf));
      num_cache_entries++;
    }
  }
  else {                                // Existing entry
    struct Cache tmp;
    memset(&tmp, 0, sizeof(struct Cache));
    tmp = Proxy_Cache[x];
    for (j=x; j<num_cache_entries; j++){
      if (j==num_cache_entries-1)
     break;
      Proxy_Cache[j] = Proxy_Cache[j+1];
    }
    Proxy_Cache[num_cache_entries -1] = tmp;
    struct tm tmp_t;
    time_t nw = time(NULL);
    tmp_t = *gmtime(&nw);
    const char* op_tmp = "%a, %d %b %Y %H:%M:%S GMT";
    strftime (Proxy_Cache[num_cache_entries - 1].Access_Date, 50,
op_tmp, &tmp_t);
  }
}

int Cache_Display () {
  int t = 0;
  if (num_cache_entries == 0)
    printf("Cache is unoccupied currently\n");
  else {
    printf("Cache count: %d\n", num_cache_entries);
    for (t=0; t<num_cache_entries; t++) {
      if (strcmp(Proxy_Cache[t].Expires, "") != 0 &&
strcmp(Proxy_Cache[t].Last_Modified, "") != 0)
        printf("Index: %d  |  URL: %s  |  Access Date: %s  |  Expires:
%s  |  Last_Modified: %s\n\n", t, Proxy_Cache[t].URL,
Proxy_Cache[t].Access_Date, Proxy_Cache[t].Expires,
Proxy_Cache[t].Last_Modified);
      else if (strcmp(Proxy_Cache[t].Expires, "") == 0 &&
strcmp(Proxy_Cache[t].Last_Modified, "") == 0)
        printf("Index: %d  |  URL: %s  |  Access Date: %s  |  Expires:
N/A  |  Last_Modified: N/A\n\n", t, Proxy_Cache[t].URL,
Proxy_Cache[t].Access_Date);
      else if (strcmp(Proxy_Cache[t].Expires, "") == 0)
        printf("Index: %d  |  URL: %s  |  Access Date: %s  |  Expires:
N/A  |  Last_Modified: %s\n\n", t, Proxy_Cache[t].URL,
Proxy_Cache[t].Access_Date, Proxy_Cache[t].Last_Modified);
      else if (strcmp(Proxy_Cache[t].Last_Modified, "") == 0)
        printf("Index: %d  |  URL: %s  |  Access Date: %s  |  Expires:
%s  |  Last_Modified: N/A\n\n", t, Proxy_Cache[t].URL,
Proxy_Cache[t].Access_Date, Proxy_Cache[t].Expires);
    }
```

```
  }
  return 0;
}


int Fresh (int cache_ptr) {
  struct tm tmp_t;
  time_t nw  = time(NULL);
  tmp_t = *gmtime(&nw);
  struct tm EXPIRES;
  if (strcmp(Proxy_Cache[cache_ptr].Expires, "") != 0) {
    strptime(Proxy_Cache[cache_ptr].Expires, "%a, %d %b %Y %H:%M:%S
%Z", &EXPIRES);
    time_t EXP = mktime(&EXPIRES);
    time_t NOW = mktime(&tmp_t);
    if (difftime (NOW, EXP) < 0)
      return 1;
    else
      return -1;
  }
  else
    return -1;
}

int Cache_Element(char *URL) {
  int b=0;
  for (b=0; b<MAX_CACHE_ENTRY; b++) {
    if (strcmp(Proxy_Cache[b].URL, URL)==0) {
      return b;
    }
  }
  return -1;
}

int WebS_Socket (char *host) {

  struct addrinfo dynamic_addr, *ai, *p;
  int ret_val = 0;
  int webs_sockfd = 0;


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

  if ((ret_val = getaddrinfo(host, "http", &dynamic_addr, &ai)) != 0)
{
    fprintf(stderr, "SERVER: %s\n", gai_strerror(ret_val));
    exit(1);
  }
  for(p = ai; p != NULL; p = p->ai_next) {
```

```
    webs_sockfd = socket(p->ai_family, p->ai_socktype, p-
>ai_protocol);
    if (webs_sockfd >= 0 && (connect(webs_sockfd, p->ai_addr, p-
>ai_addrlen) >= 0))
      break;
  }

  if (p == NULL)
    webs_sockfd = -1;

  freeaddrinfo(ai);
  return webs_sockfd;

}


int Proxy_Server(int client_fd) {
  int webs_sockfd;
  char *msg;
  char forward_client_msg[MAX_LEN] = {0};
  int ret;
  int cache_el = 0;
  //char resp[1024] = {0};
  char *resp = NULL;
  //char to_client[10240] = {0};
  char *to_client = NULL;
  //string Method;
  //string Protocol;
  char path[256];
  char hostname[64];
  int port = 80;
  char URL[256] = {0};
  char Method[8] = {0};
  char Protocol[16] = {0};
  char cond_msg[256] = {0};
  char url_parse[256] = {0};
  int check = 0;


  //memset(&url, 0, sizeof url);
  msg = (char *) malloc (MAX_LEN);
  ret = read(client_fd, msg, MAX_LEN);
  printf("SERVER: Request retrieved from client: \n%s", msg);
  if (ret < 0)
    err_sys ("SERVER: Error in extracting message request from
client");
  sscanf(msg, "%s %s %s", Method, URL, Protocol);
  //free (msg);
  //printf("SERVER: URL extracted: %s\n", URL);
  if ((cache_el = Cache_Element (URL)) != -1 && (Fresh (cache_el) ==
1)) {
    //printf("Cache_el: %d\n", cache_el);
```

```
    printf ("SERVER: Requested URL: %s is in cache and is fresh\n",
URL);
    Update_Cache(URL, NULL, 0, cache_el);
    to_client = (char *) malloc(strlen(Proxy_Cache[cache_el].body));
    memcpy(to_client, Proxy_Cache[cache_el].body,
strlen(Proxy_Cache[cache_el].body));
  }
  else {        // Either URL is not cached or it is stale
    memset(hostname, 0, 64);
    memset(path, 0, 256);
    memcpy(&url_parse[0], &URL[0], 256);
    parse_URL (url_parse, hostname, &port, path);
    if ((webs_sockfd = WebS_Socket (hostname)) == -1)
      err_sys ("SERVER: Error in connecting with web server");

    printf ("SERVER: Successfully connected to web server %d\n",
webs_sockfd);
    if (cache_el != -1) {                        // If cache entry
exists but has expired
      printf ("SERVER: Requested URL: %s is in cache but is
expired\n", URL);
      //split_URL (URL, split_url);
      if (strcmp(Proxy_Cache[cache_el].Expires, "") != 0 &&
strcmp(Proxy_Cache[cache_el].Last_Modified, "") != 0)
        snprintf(cond_msg, MAX_LEN, "%s %s %s\r\nHost: %s\r\nUser-
Agent: HTTPTool/1.0\r\nIf-Modified_Since: %s\r\n\r\n", Method, path,
Protocol, hostname, Proxy_Cache[cache_el].Expires);
      else if (strcmp(Proxy_Cache[cache_el].Expires, "") == 0 &&
strcmp(Proxy_Cache[cache_el].Last_Modified, "") == 0)
        snprintf(cond_msg, MAX_LEN, "%s %s %s\r\nHost: %s\r\nUser-
Agent: HTTPTool/1.0\r\nIf-Modified_Since: %s\r\n\r\n", Method, path,
Protocol, hostname, Proxy_Cache[cache_el].Access_Date);
      else if (strcmp(Proxy_Cache[cache_el].Expires, "") == 0)
        snprintf(cond_msg, MAX_LEN, "%s %s %s\r\nHost: %s\r\nUser-
Agent: HTTPTool/1.0\r\nIf-Modified_Since: %s\r\n\r\n", Method, path,
Protocol, hostname, Proxy_Cache[cache_el].Last_Modified);
      else if (strcmp(Proxy_Cache[cache_el].Last_Modified, "") == 0)
        snprintf(cond_msg, MAX_LEN, "%s %s %s\r\nHost: %s\r\nUser-
Agent: HTTPTool/1.0\r\nIf-Modified_Since: %s\r\n\r\n", Method, path,
Protocol, hostname, Proxy_Cache[cache_el].Expires);
      printf("Conditional GET Generated: \n%s", cond_msg);
      write(webs_sockfd, cond_msg, MAX_LEN);
      //resp = malloc (10240);    // FIXME: May be needed to increase
allocation
      //memset(resp, 0, 1024);
      resp = (char *) malloc (100000);
      check = Extract_Read(webs_sockfd, resp);
      //printf("Checking: %d\n", check);
      //Extract_Read(webs_sockfd, resp);
      to_client = (char *) malloc(strlen(resp));
      if (strstr(resp, "304 Not Modified") != NULL) {
```

```
        printf("'304 Not Modified' received. Sending file in
cache\n");
        memcpy(to_client, Proxy_Cache[cache_el].body,
strlen(Proxy_Cache[cache_el].body));
      Update_Cache(URL, NULL, 0, cache_el);
       }
       else {
         printf("SERVER: File was modified\n");
         memcpy(to_client, resp, strlen(resp));
         Update_Cache(URL, NULL, 0, cache_el);         // move to head
(LRU) of the queue
         Proxy_Cache[--num_cache_entries] = Clear_Entry;
// Popping LRU
         Update_Cache(URL, resp, 1, 0);     // treat like a new entry
as it was modified
       }
     }
     else {             // document is not cached
       printf ("SERVER: Requested URL is not in cache\n");
       memset(forward_client_msg, 0, MAX_LEN);
       snprintf(forward_client_msg, MAX_LEN, "%s %s %s\r\nHost:
%s\r\nUser-Agent: HTTPTool/1.0\r\n\r\n", Method, path, Protocol,
hostname);
       printf("SERVER: Request generated: \n%s", forward_client_msg);
       write(webs_sockfd, forward_client_msg, MAX_LEN);
       resp = (char *) malloc (100000);
       check = Extract_Read(webs_sockfd, resp);
       to_client = (char *) malloc(strlen(resp));
       memcpy(to_client, resp, strlen(resp));
       Update_Cache(URL, resp, 1, 0);
     }
   }
   Cache_Display();
   write(client_fd, to_client, strlen(to_client) + 1);
}

int main (int argc, char *argv[])
{
   int sockfd, comm_fd, bind_fd, listen_fd;
   int port_number ;
   struct sockaddr_storage remoteaddr;
   socklen_t addrlen;

   if (argc != 3){
     err_sys ("USAGE: ./proxy <Server IP Address> <Port_Number>");
     return 0;
   }

   port_number = atoi(argv[2]);
   struct sockaddr_in servaddr;
   sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

```
  if (sockfd < 0)
    err_sys ("ERR: Socket Error");

  memset( &servaddr, 0 , sizeof(servaddr));
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = inet_addr(argv[1]);
  servaddr.sin_port = htons(port_number);
  bind_fd = bind(sockfd, (struct sockaddr *)&servaddr,
sizeof(servaddr));
  if (bind_fd < 0)
    err_sys ("ERR: Bind Error");
  listen_fd = listen(sockfd, 10);
  if (listen_fd < 0)
    err_sys ("ERR: Listen Error");
  memset(Proxy_Cache,0,MAX_CACHE_ENTRY*sizeof(struct Cache));

  addrlen = sizeof remoteaddr;
  pthread_t x;
  printf("\nPROXY SERVER is online\n\n");
  while(1)
  {
    comm_fd = accept(sockfd, (struct sockaddr*)&remoteaddr,&addrlen);
    pthread_create(&x, NULL, (void *)(&Proxy_Server), (void
*)(intptr_t)comm_fd);
  }

}
```

CLIENT :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <errno.h>
#include <dirent.h>
#include <pthread.h>
#include <signal.h>
#include <sys/wait.h>

int parse_URL (char* URL, char *hostname, int *port, char *path) {
  char *token;
  char *host_temp, *path_temp;
  char *tmp1, *tmp2;
  int num = 0;
  char s[16];
  if (strstr(URL,"http") != NULL){
```

```
    token = strtok(URL, ":");
    tmp1 = token + 7;
  }
  else{
    tmp1 = URL;
  }
  tmp2 = malloc (64);
  memcpy(tmp2, tmp1, 64);
  if(strstr(tmp1, ":") != NULL){
    host_temp = strtok(tmp1, ":");
    *port = atoi(tmp1 + strlen(host_temp) + 1);
    sprintf(s, "%d", *port);
    path_temp = tmp1 + strlen(host_temp) + strlen(s) + 1;
  }
  else{
    host_temp = strtok(tmp1, "/");
    *port = 80;
    path_temp = tmp2 + strlen(host_temp);
  }
  if (strcmp(path_temp, "") == 0)
    strcpy(path_temp, "/");
  memcpy(hostname, host_temp, 64);
  memcpy(path, path_temp, 256);
  return(0);
}



int err_sys(const char* x)    // Error display source code
{
  perror(x);
  exit(1);
}


int main(int argc,char *argv[])
{
  int sockfd, inet_a2n_ret, conn_ret, n;
  char buff[100000] = {0};
  int sendret = 0;
  int recvret = 0;
  unsigned int port_number ;
  char *p, *ptr;
  char req[100];
  char path[256] = {0};
  char hostname[64] = {0};
  int port = 80;
  char URL[256] = {0};

  if (argc != 4){
```

```
    err_sys ("USAGE: ./client <Server_IP_Address> <Port_Number>
<URL>");
    exit(1);
  }

  port_number = atoi(argv[2]);

  struct sockaddr_in servaddr;

  bzero(&servaddr,sizeof servaddr);
  servaddr.sin_family=AF_INET;
  servaddr.sin_port=htons(port_number);
  inet_a2n_ret = inet_aton(argv[1], (struct in_addr
*)&servaddr.sin_addr.s_addr);      // FIXME: Maybe a problem
  if (inet_a2n_ret <= 0)
    err_sys ("ERR: inet_aton error");

  sockfd=socket(AF_INET,SOCK_STREAM,0);
  if (sockfd < 0)
    err_sys ("ERR: Socket Error");

  conn_ret = connect(sockfd,(struct sockaddr
*)&servaddr,sizeof(servaddr));
  if (conn_ret < 0)
    err_sys ("ERR: Connect Error");
  memset(req, 0, 100);
  sprintf(req, "GET %s HTTP/1.0\r\n", argv[3]);
  printf("Request sent to proxy server: \n%s\n", req);
  sendret = send(sockfd, req, strlen(req), 0);
  if (sendret == -1) {
      err_sys("CLIENT: Send");
      exit(2);
  }
  memset(buff, 0, 100000);
  parse_URL(argv[3], hostname, &port, path);
  FILE *fp;
  fp=fopen(hostname, "w");
  printf("Waiting for response\n");
  recvret = recv(sockfd, buff, 100000, 0);
  if (recvret <= 0) {
    err_sys("CLIENT: Recv");
    fclose(fp);
    close(sockfd);
    return 1;
  }
  if((strstr(buff, "200")) != NULL)
    printf("'200 OK' received. Saving to file: %s\n", hostname);
  else if ((strstr(buff, "400") != NULL))
    printf("'400 Bad Request' received. Saving to file: %s\n",
hostname);
  else if ((strstr(buff, "404") != NULL))
```

```
    printf("'404 Page Not Found' received. Saving to file: %s\n",
hostname);
  ptr = strstr(buff, "\r\n\r\n");
  fwrite(ptr+4, 1, strlen(ptr)-4, fp);
  fclose(fp);
  close(sockfd);

  return 0;

}
```

MAKEFILE :

```
# 'make all' for compiling all code in package
all: proxy client

# 'make server' for compiling Server.c
proxy: Proxy.c
     gcc -I . -pthread Proxy.c -o proxy

# 'make client' for compiling Client.c
client: Client.c
     gcc -I . Client.c -o client

# 'make clean' for discarding all previously created object files
clean:
     $(RM) client server
```