COMPUTER NETWORKS(CS425A)

PROJECT 2:HTTP PROXY

31st August, 2016

KRITI JOSHI - 13358

kritij@iitk.ac.in

IMPLEMENTED OPTIONS:

1) An HTTP-Proxy server to handle “GET” request from client.
2) Can specify port-number from command line.
3) 500 Internal Server Error reported for memory overflow, bad requests and methods other than “GET”.
4) Connection attribute set to “close” and HTTP version set to “1.0” while sending request to server.
5) Concurrent requests handled by forking children.

DESIGN DECISIONS:

1) Forking : On getting a new request, the parent forks a child which handles the request once and dies after closing the socket. The maximum number of children maintained in the assignment is 30 (otherwise program gets extremely slow on large number of concurrent reads). The count of number of children is maintained using **share memory.**

2) BUFFER_SIZE: For the assignment, mostly firefox has been used as a client which has max REQUEST size of 8 KB. The BUFER SIZE is set accordingly.

TESTS CONDUCTED: Tests

1) File: proxy_tester.py

```
kriti@hp:Project2$ python proxy_tester.py proxy 9090
Binary: proxy
Running on port 9090
Listening...
### Testing: http://example.com/
http://example.com/: [PASSED]

### Testing: http://sns.cs.princeton.edu/
http://sns.cs.princeton.edu/: [PASSED]

### Testing: http://www.cs.princeton.edu/people/faculty
http://www.cs.princeton.edu/people/faculty: [PASSED]

### Testing: http://www.cs.princeton.edu/sites/default/files/styles/gallery_full/public/gallery-images/CS_building9.jpg?itok=meb0LzhS
http://www.cs.princeton.edu/sites/default/files/styles/gallery_full/public/gallery-images/CS_building9.jpg?itok=meb0LzhS: [PASSED]

Summary:
        4 of 4 tests passed.
kriti@hp:Project2$
```

2) File: proxy_tester_conc.py:

```
    100%   1061 (longest request)
http://example.com/ with args -n 200 -c 10: [PASSED]

### Testing apache benchmark on args [-n 1000 -c 50]
    This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
    Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
    Licensed to The Apache Software Foundation, http://www.apache.org/

    Benchmarking example.com [through 127.0.0.1:9090] (be patient)
    Completed 100 requests
    Completed 200 requests
    Completed 300 requests
    Completed 400 requests
    Completed 500 requests
    Completed 600 requests
    Completed 700 requests
    Completed 800 requests
    Completed 900 requests
    Completed 1000 requests
    Finished 1000 requests


    Server Software:        ECS
    Server Hostname:        example.com
    Server Port:            80

    Document Path:          /
    Document Length:        1270 bytes

    Concurrency Level:      50
    Time taken for tests:   7.422 seconds
    Complete requests:      1000
    Failed requests:        0
    Total transferred:      1704000 bytes
    HTML transferred:       1270000 bytes
    Requests per second:    134.74 [#/sec] (mean)
    Time per request:       371.092 [ms] (mean)
    Time per request:       7.422 [ms] (mean, across all concurrent requests)
    Transfer rate:          224.21 [Kbytes/sec] received

    Connection Times (ms)
    min  mean[+/-sd] median   max
    Connect:        0   69 286.7      0    3003
    Processing:    19  231 357.6    172    6422
    Waiting:       19  230 357.3    171    6421
    Total:         20  300 504.5    175    7421

    Percentage of the requests served within a certain time (ms)
    50%    175
    66%    205
    75%    230
    80%    248
    90%    596
    95%   1202
    98%   1528
    99%   2042
    100%   7421 (longest request)
http://example.com/ with args -n 1000 -c 50: [PASSED]

Summary:
        Type multi-process: 13 of 13 tests passed.
kriti@hp:Project2$
```

APPENDIX:

SOURCE CODE:

```
#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <time.h>

#include <bits/stdc++.h>

#include <dirent.h>

#include <netinet/tcp.h>

// To check whether file or directory

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

// Fork child

#include <unistd.h>

// Parsing
```

```
#include "proxy_parse.h"
#include <netdb.h>
#include <sys/wait.h>
#include <sys/mman.h>

#define BUFFER_SIZE 8192
#define SMALL 30
#define VERY_SMALL 10
#define MAX 30

using namespace std;

/*
   Function: Handles client's request
*/
void serveClient(int clientFileDescriptor){
   char input[BUFFER_SIZE];
   int socketFileDescriptor ;

   // Receive request from client
   bzero(input,BUFFER_SIZE);
```

```
int bytesReadNow;
int tillNow = 0;



char* endIndex;
do{
    bytesReadNow = recv(clientFileDescriptor,input+tillNow,BUFFER_SIZE,0);
    endIndex = strstr(input, "\r\n\r\n");
    tillNow+=bytesReadNow;
    if (bytesReadNow == 0){
        break;
    }
    if (bytesReadNow < 0){
        printf("Error in connection.\n");
        return;
    }
}while(endIndex==NULL);

// Client doesn't send request
if(bytesReadNow<=0){
    return;
```

```
    }


    // Parse request
    ParsedRequest *req = ParsedRequest_create();
    if(ParsedRequest_parse(req,input,strlen(input))<0){
        printf("parse failed\n");
        char reply[BUFFER_SIZE];
        // goto label;
        sprintf(reply, "HTTP/1.0 500 Internal Server Error\r\nContent-Type:
text/html\r\n\r\n");
        //send reply
        int len = strlen(reply);
        send(clientFileDescriptor,reply,len,0);
        return;
    }
    if (ParsedHeader_set(req, "Host", req->host) < 0){
     printf("set header key not work\n");
     return;


    }
```

```
if (ParsedHeader_set(req, "Connection", "close") < 0){
 printf("set header key not work\n");
 return;

}
// Default port 80
if(req->port==NULL){
    req->port = new char[4];
    sprintf(req->port,"80");
}

// set http version to 1.0
if(strcmp(req->version,"HTTP/1.0")){
    // cout<<"Did version change  from :"<<req->version<<endl;
    sprintf(req->version,"HTTP/1.0");
}

socketFileDescriptor = socket(AF_INET, SOCK_STREAM, 0);

if(socketFileDescriptor<0){
    cout<<"Couldn't create socket\n";
```

```
        return;
    }

    // Server address information
    struct sockaddr_in serverAddress;
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(atoi(req->port));

    // Get IP
    struct in_addr **addr_list = (struct in_addr **) gethostbyname(req->host)->h_addr_list;
    for(int i = 0; addr_list[i] != NULL; i++){
        serverAddress.sin_addr = *addr_list[i];
        // cout<<req->host<<" resolved to "<<inet_ntoa(*addr_list[i])<<endl;
        break;
    }

    // Connect to server
    int status= connect(socketFileDescriptor, (struct sockaddr*)&serverAddress,
sizeof(serverAddress));
    if(status != 0){
      fprintf(stderr,"Trying to connect\n");
```

```
}

// Build request for server
int rlen = ParsedRequest_totalLen(req);
char *b = (char *)malloc(rlen+1);
bzero(b,rlen);
if (ParsedRequest_unparse(req, b, rlen) < 0) {
    printf("unparse failed\n");
}
b[rlen]='\0';

// Send request to server
int bytesSent = send(socketFileDescriptor,b,strlen(b),0);
// cout<<"Request sent to server: "<<bytesSent<<endl;
// cout<<b<<"\n-----------------------------------\n";
free(b);

// Receive response from server and send it to client
char serverResponse[BUFFER_SIZE];
bytesSent=0;
bzero(serverResponse,BUFFER_SIZE);
```

```
    bytesReadNow = recv(socketFileDescriptor,&serverResponse,BUFFER_SIZE,0);
    // cout<<"Read: "<<bytesReadNow<<endl;
    while(bytesReadNow>0){
        bytesSent += send(clientFileDescriptor,serverResponse,bytesReadNow,0);
        // cout<<"Sent: "<<bytesSent<<endl;
        bzero(serverResponse,bytesReadNow);
        bytesReadNow = recv(socketFileDescriptor,&serverResponse,BUFFER_SIZE,0);
    }

    close(socketFileDescriptor);
}
static int* numChild;
int main(int argc, char *argv[]){

    if(argc!=2){
        cout<<"Please enter port-number\n";
        return 1;
    }

    // Create Socket
    int socketFileDescriptor = socket(AF_INET, SOCK_STREAM, 0);
```

```
    int enable = 1;
    if (setsockopt(socketFileDescriptor, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int))
< 0)
        cout<<"setsockopt(SO_REUSEADDR) failed\n";

    if(socketFileDescriptor<0){
        cout<<"Couldn't create socket\n";
        return 1;
    }

    //server Address information
    struct sockaddr_in serverAddress;
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);              //binds to all available
interfaces
    serverAddress.sin_port = htons(atoi(argv[1]));

    //Assign socket address to declare socket
    int fails=bind(socketFileDescriptor, (struct sockaddr*)&serverAddress,
sizeof(serverAddress));
    if(fails){
        fprintf(stderr,"Couldn't bind to the port: %d\n",atoi(argv[1]));
```

```
    return 1;
}

//Start listening on the port, maximum allowed clients is 20
fails=listen(socketFileDescriptor,5);
if(!fails){
    cout<<"Listening...\n";
}

numChild = (int*)mmap(NULL, sizeof *numChild, PROT_READ | PROT_WRITE,
            MAP_SHARED | MAP_ANONYMOUS, -1, 0);

bool done = false;
int status;

// Fork children on connection
while(!done){
    int clientFileDescriptor = accept(socketFileDescriptor,NULL,NULL);
    while(*numChild>=MAX){
        wait(&status);
    }
```

```
        int pid = fork();
        if(pid==0){
            *numChild = *numChild + 1;
            serveClient(clientFileDescriptor);
            close(clientFileDescriptor);
            done = true;
            *numChild = *numChild - 1;
        }
        else if(pid<0){
            cout<<"unable to fork\n";
            return 1;
        }else{
            close(clientFileDescriptor);
        }
    }

    return 0;
}
```