**Lab Assignment-3 Contents**

**Kanti**

**Gnanadeep**

**Usha**

**Web server**

**Readme**

**Implementation of Web server  
  
1. To open an editor  
gedit http\_server.c  
  
2. To compile the server  
make all  
  
3. To execute the server program and start the server  
./server 127.0.0.1 PORT**

**Make file**

.PHONY: server

all: server

server:

gcc http\_server.c -o server

clean:

rm server

**Server Source file**

// CMPE\_297 Assignment\_3

// WEB Server

// Author - Team #2

// This WEB server program connects with TCP clients

// accepts HTTP 1.0 GET requests from client. processes

// it and searches for the requested file in the entire

// system. If found HTTP response status + HEADER + file

// is sent to the client. If file is not found then an

// error is sent back and closes the connection. waits for

// the next connection.

// To compile: gcc http\_server.c -o server

// To run: ./server <IP address> <server\_port>

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<string.h>

#include<netinet/in.h>

#include<time.h>

#include<dirent.h>

#include<netdb.h>

#include<unistd.h>

#include<arpa/inet.h>

#include <dirent.h>

#include <errno.h>

#include <sys/stat.h>

#define BUF\_SIZE 1024

#define CLADDR\_LEN 100

int createSocket(char \* host, int port);

int listenForRequest(int sockfd);

char \* getFileType(char \* file);

int Find(char \*\*\* list, char \* file\_name, char \* directory, int \* location);

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

DIR \* dirptr;

FILE \* fileptr;

time\_t timenow;

struct tm \* timeinfo;

time (&timenow);

timeinfo = localtime(&timenow);

char \* header, \* request, \* path, \* filename\_requested, \* host;

char \* dir, \* temp;

int port, sockfd, connfd;

char get[3], http[9];

char filepath[BUF\_SIZE];

char http\_not\_found[] = "HTTP/1.0 404 Not Found\n";

char http\_ok[] = "HTTP/1.0 200 OK\n";

char buffer[BUF\_SIZE];

char \* contentType;

if (argc != 3) {

printf("usage: [host] [portnumber]\n");

exit(1);

}

header = (char\*)malloc(BUF\_SIZE\*sizeof(char));

request = (char\*)malloc(BUF\_SIZE\*sizeof(char));

path = (char\*)malloc(BUF\_SIZE\*sizeof(char));

filename\_requested = (char\*)malloc(BUF\_SIZE\*sizeof(char));

char \*\* found\_list = (char\*\*)malloc(sizeof(char\*) \* 30);

if (!found\_list) return 0;

int counter\_location = 0;

char \* ini\_dir = (char\*)malloc(sizeof(char) \* 500);

if (!ini\_dir) {

free (found\_list);

return 0;

}

host = argv[1];

port = atoi(argv[2]);

sockfd = createSocket(host, port);

for (;;) {

memset(ini\_dir, 0, sizeof(ini\_dir));

printf("--------------------------------------------------------\n");

printf("Waiting for a connection...\n");

connfd = listenForRequest(sockfd);

//gets the request from the connection

recv(connfd, request, 100, 0);

printf("Processing request...\n");

//parses request

sscanf(request, "%s %s %s", get, path, http);

filename\_requested = path + 1; //ignores the first slash

//search the filename\_requested

Find(&found\_list, filename\_requested, ini\_dir, &counter\_location);

if (counter\_location <= 0) {

printf("File not found!\n");

send(connfd, http\_not\_found, strlen(http\_not\_found), 0); //sends HTTP 404

sprintf(header, "Content-Type: application/text\nContent-Length: 0\n\n");

send(connfd, header, strlen(header), 0); //sends the header

close(connfd);

continue;

}

strncpy(filepath, found\_list[counter\_location-1], strlen(found\_list[counter\_location-1]));

filepath[strlen(found\_list[counter\_location-1])] = '\0';

printf("\nSERVER filepath: %s ", filepath);

// contentType = getFileType(filename\_requested);

contentType = "application/octet-stream";

//sprintf(header, "Date: %sHostname: %s:%d\nLocation: %s\nContent-Type: %s\nContent-Length: %d\n\n", asctime(timeinfo), host, port, filename\_requested, contentType, 1053);

struct stat statBuf = {0};

if (0 > stat(filepath, &statBuf)) {

printf("Failed to get stat of file\n");

return 1;

}

int file\_length = statBuf.st\_size;

sprintf(header, "Content-Type: application/text\nContent-Length: %d\n\n", file\_length);

if ((fileptr = fopen(filepath, "r")) == NULL ) {

} else {

printf("Sending the file...\n");

send(connfd, http\_ok, strlen(http\_ok), 0); //sends HTTP 200 OK

recv(connfd, buffer, BUF\_SIZE, 0);

printf("Received %s\n", buffer);

if ((temp = strstr(buffer, "OK")) == NULL) {

printf("Operation aborted by the user!\n");

//break;

}

send(connfd, header, strlen(header), 0); //sends the header

memset(&buffer, 0, sizeof(buffer));

while (!feof(fileptr)) { //sends the file

int readBytes = fread(&buffer, 1, sizeof(buffer), fileptr);

printf("Read %d bytes\n", readBytes);

send(connfd, buffer, readBytes, 0);

memset(&buffer, 0, sizeof(buffer));

}

printf("File sent...\n");

}

printf("Processing completed...\n");

close(connfd);

}

close(sockfd);

free(header);

free(request);

free(path);

free(filename\_requested);

if (counter\_location > 0) {

while (--counter\_location >= 0) {

free (found\_list[counter\_location]);

}

}

return 0;

}

// Function to Create TCP socket end point

int createSocket(char \* host, int port) {

int sockfd;

struct sockaddr\_in addr;

struct hostent \* host\_ent;

char \* hostAddr;

memset(&addr, 0, sizeof(addr));

addr.sin\_family = AF\_INET;

addr.sin\_addr.s\_addr = INADDR\_ANY;

addr.sin\_port = htons((short)port);

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0) {

printf("Error creating socket!\n");

exit(1);

}

printf("Socket created...\n");

if (bind(sockfd, (struct sockaddr \*)&addr, sizeof(addr)) < 0) {

printf("Error binding socket to port!\n");

exit(1);

}

printf("Binding done...\n");

return sockfd;

}

// Function to listen for incoming requests

int listenForRequest(int sockfd) {

int conn;

char hostip[32];

struct sockaddr\_in addr;

struct hostent \* host;

struct in\_addr inAddr;

int len;

addr.sin\_family = AF\_INET;

listen(sockfd, 5); //maximum 5 connections

len = sizeof(addr);

if ((conn = accept(sockfd, (struct sockaddr \*)&addr, &len)) < 0) {

printf("Error accepting connection!\n");

exit(1);

}

printf("Connection accepted...\n");

inet\_ntop(AF\_INET, &(addr.sin\_addr), hostip, 32);

inet\_pton(AF\_INET, hostip, &inAddr);

host = gethostbyaddr(&inAddr, sizeof(inAddr), AF\_INET);

printf("---Connection received from: %s [IP= %s]---\n", host->h\_name, hostip);

return conn;

}

// Function find type of the file

// HTTP client is requesting

char \* getFileType(char \* file) {

char \* temp;

if ((temp = strstr(file, ".html")) != NULL) {

return "text/html";

} else if ((temp = strstr(file, ".pdf")) != NULL) {

return "application/pdf";

} else if ((temp = strstr(file, ".txt")) != NULL) {

return "text/html";

}

}

// Function to search a file system wide.

int Find(char \*\*\* list, char \* file\_name, char \* directory, int \* location)

{

DIR \* opened\_dir;

struct dirent \* directory\_structure = NULL;

struct dirent entry;

char \* temp\_dir = (char\*)malloc(sizeof(char) \* 300);

if (!temp\_dir) return -1;

strcat(directory, "/");

//printf("Find:directory = %s\n", directory);

opened\_dir = opendir(directory);

if (opened\_dir == NULL) {

free (temp\_dir);

return -1;

}

while (1) {

directory\_structure = readdir(opened\_dir);

if (!directory\_structure) {

closedir(opened\_dir);

free (temp\_dir);

return 0;

}

if (!strcmp(directory\_structure->d\_name, "..") ||

!strcmp(directory\_structure->d\_name, ".")) continue;

else if (!strcmp(directory\_structure->d\_name, file\_name)) {

(\*list)[\*location] = (char\*)malloc(sizeof(char) \* 300);

sprintf((\*list)[\*location], "%s%s", directory, directory\_structure->d\_name);

(\*location)++;

}

else if (directory\_structure->d\_type == DT\_DIR) {

sprintf(temp\_dir, "%s%s", directory, directory\_structure->d\_name);

Find(list, file\_name, temp\_dir, location);

}

}

return 0;

}

**Output:**

kanti@kanti-linux:~/kanti\_sem\_3/207/Assgn\_3/HTTP\_CLIENT\_SERVER$ ./server 127.0.0.1 8500

Socket created...

Binding done...

--------------------------------------------------------

Waiting for a connection...

Connection accepted...

---Connection received from: localhost [IP= 127.0.0.1]---

Processing request...

SERVER filepath: /home/kanti/Downloads/one\_k\_file.txt Sending the file...

Received OK

File sent...

Processing completed...

--------------------------------------------------------

Waiting for a connection...

**Web Client**

**Readme**

**Implementation of Web client  
  
1. To open an editor  
gedit http\_client.c  
  
2. To compile the client  
make all  
  
3.  
a. To execute the client program on the console  
./client 127.0.0.1/FILE\_NAME PORT  
  
b. To execute the client on the browser with standard web browser  
127.0.0.1:PORT/FILE\_NAME  
  
c. TO execute the client with standard server  
./client URL 80**

**Make file**

.PHONY: client

all: client

client:

gcc http\_client.c -o client

clean:

rm client

**Server Source file**

// CMPE\_297 Assignment\_3

// WEB Client

// Author - Team #2

// This WEB client program connects to the WEB server

// sends a HTTP1.0 GET request for a text file. If the

// requested file exists on the server, Server responds

// with HTTP1.0 response header and the requested file.

// Other wise client receives the error message.

// To compile: gcc http\_client.c -o client

// To run: ./client <server\_address/file\_name> <server\_port>

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<string.h>

#include<ctype.h>

#include<arpa/inet.h>

#include<netdb.h>

#include<time.h>

#define BUF\_SIZE 1024

int connectsock(const char \*host, const char \*service, const char \*transport);

void split\_url(char\* url, char\*\* host);

int get\_request(char \* url, char \* port);

int isValidIP(char \* ip);

int parseHeader(char \* header);

char\* splitKeyValue(char \* line, int index);

void openFile();

long GetFileSize(const char\* filename);

FILE \* fileptr;

//char keys[][25] = {"Date: ", "Hostname: ", "Location: ", "Content-Type: "};

char keys[][25] = {"Content-Type: ", "Content-Length: "};

char status[4] = {0, 0, 0, 0};

char contentFileType[100];

char path[1000];

char getrequest[1024];

// Main Function

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

{

int sock, ret;

char\* host = NULL;

char \* url, \* temp;

char \* fileName;

char status\_ok[] = "OK";

char buffer[BUF\_SIZE];

char http\_not\_found[] = "HTTP/1.0 404 Not Found";

char http\_ok[] = "HTTP/1.0 200 OK";

char location[] = "Location: ";

char contentType[] = "Content-Type: ";

int sPos, ePos;

clock\_t start,stop;

double result1;

if(argc < 3)

{

printf("Usage: ./client <URL> <server port> \n");

printf("Example Usage: ./client 127.0.0.1/one\_k\_file.txt 4000 \n");

exit(1);

} else

{

printf("\n>> TCP Web Test Client. This program send HTTP1.0 GET request for a\n");

printf(">> file to the TCP Web server. File is saved in the current directory. \n\n");

url = argv[1];

//split argv[1] into server address and filename arguments

split\_url(argv[1], &host);

printf("CLIENT: Sending getrequest = %s\n", getrequest);

//create tcp socket end point

sock = connectsock(host, argv[2], "tcp");

start=clock();

// Send HTTP 1.0 GET request over TCP connection

write(sock, getrequest, strlen(getrequest));

// handle HTTP response status from the server

memset(&buffer, 0, sizeof(buffer));

ret = recv(sock, buffer, BUF\_SIZE, 0);

if (ret < 0) {

printf("Error receiving HTTP status!\n");

} else {

printf("RESPONSE: HTTP STATUS FROM SERVER (STATUS SIZE: %ld bytes.) :\n", (strlen(buffer)-1));

printf("%s\n", buffer);

if ((temp = strstr(buffer, http\_ok)) != NULL) {

send(sock, status\_ok, strlen(status\_ok), 0);

} else {

close(sock);

return 0;

}

}

// handle HTTP response header

memset(&buffer, 0, sizeof(buffer));

ret = recv(sock, buffer, BUF\_SIZE, 0);

if (ret < 0) {

printf("Error receiving HTTP header!\n");

} else {

printf("RESPONSE: HTTP HEADER FROM SERVER (HEADER SIZE: %ld bytes.) :\n", (strlen(buffer)-1));

printf("%s\n", buffer);

if (parseHeader(buffer) == 0) {

send(sock, status\_ok, strlen(status\_ok), 0);

} else {

printf("Error in HTTP header!\n");

close(sock);

return 0;

}

}

// Open file for writing

fileptr = fopen(path, "w");

if (fileptr == NULL) {

printf("Error opening file!\n");

close(sock);

return 0;

}

// Handle file transfer from the server

memset(&buffer, 0, sizeof(buffer));

while (1) {

int recvBytes = recv(sock, buffer, BUF\_SIZE-1, 0);

if (recvBytes <= 0)

break;

printf("Recieved %d bytes\n", recvBytes);

if ((strstr(contentFileType, "application/html")) != NULL) {

printf("Wrote %d bytes\n", fprintf(fileptr, "%s", buffer));

} else {

fwrite(&buffer, recvBytes, 1, fileptr);

}

memset(&buffer, 0, sizeof(buffer));

}

stop=clock();

printf("File Transfer successful. File stored locally on the disk. \n");

fclose(fileptr);

close(sock);

//openFile();

printf("RESPONSE: FILE SIZE: %ld bytes.\n", GetFileSize(path));

result1=((double)(stop-start))/CLOCKS\_PER\_SEC;

printf("Time required to fetch the file is %f seconds.\n",result1);

return 0;

}

}

// Function to Handle TCP connection request

int connectsock(const char \*host, const char \*service, const char \*transport)

{

struct hostent \*phe;

struct servent \*pse;

struct protoent \*ppe;

struct sockaddr\_in remote\_server;

int sock, type;

//reset socket address structure.

memset(&remote\_server,0,sizeof(remote\_server));

remote\_server.sin\_family = AF\_INET;

//map service name to port number

if(pse = getservbyname(service, transport))

remote\_server.sin\_port = pse->s\_port;

else if((remote\_server.sin\_port = htons((u\_short)atoi(service))) == 0)

//perror("Cant get \"%s\" service entry\n", service);

perror("Cant get service entry\n");

//map host name to IP address allowing dotted decimal

if(phe = gethostbyname(host))

memcpy(&remote\_server.sin\_addr, phe->h\_addr, phe->h\_length);

else if ((remote\_server.sin\_addr.s\_addr = inet\_addr(host) == INADDR\_NONE))

//perror("Cant get \"%s\" host entry\n", host);

perror("Cant get host entry\n");

bzero(&remote\_server.sin\_zero, 8);

//map transport protocol name to protocol number

ppe = getprotobyname(transport);

if((ppe == NULL))

//perror("cant get \"%s\" protocol entry\n", transport);

perror("Cant get protocol entry\n");

//use protocol to chose a socket type

if(strcmp(transport, "udp") == 0)

type = SOCK\_DGRAM;

else

type = SOCK\_STREAM;

//allocate a socket

sock = socket(PF\_INET, type, ppe->p\_proto);

if(sock < 0)

//perror("Cant create socket: %s\n", strerror(errno));

perror("Cant create socket\n");

//connect the socket

if(connect(sock, (struct sockaddr \*)&remote\_server, sizeof(remote\_server)) < 0)

//perror("Cant connect to %s %s: %s\n", host, service, sererror(errno));

perror("Cant connect to the remote host\n");

return sock;

}

// Function to split url into host and filenames

void split\_url(char\* url, char\*\* host)

{

char\* ptr;

char\* temp\_host, \*host1;

if (isValidIP(url)) { //when an IP address is given

sprintf(getrequest, "GET / HTTP/1.0\nHOST: %s\n\n", url);

} else { //when a host name is given

if ((ptr = strstr(url, "/")) == NULL) {

//when hostname does not contain a slash

sprintf(getrequest, "GET / HTTP/1.0\nHOST: %s\n\n", url);

} else {

//when hostname contains a slash, it is a path to file

strcpy(path, ptr);

\*host = strtok(url, "/");

sprintf(getrequest, "GET %s HTTP/1.0\nHOST: %s\n\n", path, url);

}

}

ptr = strtok(path, "/");

strcpy(path, ptr);

}

// Function to validate given IP address

int isValidIP(char \* ip) {

struct sockaddr\_in addr;

int valid = inet\_pton(AF\_INET, ip, &(addr.sin\_addr));

return valid != 0;

}

// Functio to parse HTTP header

int parseHeader(char \* header) {

//"Date: %sHostname: %s:%d\nLocation: %s\nContent-Type: %s\n\n"

char \* line, \* key, \* value;

char temp[100];

int i = 0;

line = strtok(header, "\n");

while (line != NULL) {

//printf("%s\n", line);

strcpy(temp, line);

value = splitKeyValue(line, i);

if (i == 3) {

strcpy(contentFileType, value);

}

//printf("value=%s\n", value);

line = strtok(NULL, "\n");

i++;

}

for (i = 0; i < 2; i++) {

if (status[i] == 0) return 1;

//printf("status[%d]=%d\n", i, status[i]);

}

return 0;

}

// Helper function

char \* splitKeyValue(char \* line, int index) {

char \* temp;

if ((temp = strstr(line, keys[index])) != NULL) {

temp = temp + strlen(keys[index]);

status[index] = 1;

}

return temp;

}

// Function to open the transfered file

void openFile() {

char \* temp;

char command[100];

char fileName[1000];

strcpy(fileName, path);

//printf("File Name: %s\n", fileName);

//printf("Content Type: %s\n", contentFileType);

if ((temp = strstr(contentFileType, "text/html")) != NULL) {

if ((temp = strstr(fileName, ".txt")) != NULL) {

sprintf(command, "gedit %s", fileName);

} else {

sprintf(command, "firefox %s", fileName);

}

system(command);

} else if ((temp = strstr(contentFileType, "application/pdf")) != NULL) {

sprintf(command, "acroread %s", fileName);

system(command);

} else {

printf("The filetype %s is not supported. Failed to open %s!\n", contentFileType, fileName);

}

}

long GetFileSize(const char\* filename)

{

long size;

FILE \*f;

//printf("getfilesize: filename = %s\n", filename);

f = fopen(filename, "rb");

if (f == NULL){

return -1;

printf("error...\n");

}

fseek(f, 0, SEEK\_END);

size = ftell(f);

fclose(f);

//printf("Size in getfilesize: %ld\n", size);

return size;

}

**Output:**

kanti@kanti-linux:~/kanti\_sem\_3/207/Assgn\_3/HTTP\_CLIENT\_SERVER/test\_client$ ./client 127.0.0.1/one\_k\_file.txt 8500

>> TCP Web Test Client. This program send HTTP1.0 GET request for a

>> file to the TCP Web server. File is saved in the current directory.

CLIENT: Sending getrequest = GET /one\_k\_file.txt HTTP/1.0

HOST: 127.0.0.1

RESPONSE: HTTP STATUS FROM SERVER (STATUS SIZE: 15 bytes.) :

HTTP/1.0 200 OK

RESPONSE: HTTP HEADER FROM SERVER (HEADER SIZE: 52 bytes.) :

Content-Type: application/text

Content-Length: 1053

Recieved 1023 bytes

Recieved 1 bytes

File Transfer successful. File stored locally on the disk.

RESPONSE: FILE SIZE: 1024 bytes.

Time required to fetch the file is 0.000230 seconds.

kanti@kanti-linux:~/kanti\_sem\_3/207/Assgn\_3/HTTP\_CLIENT\_SERVER/test\_client$ ls

client http\_client.c http\_client\_old.c one\_k\_file.txt

kanti@kanti-linux:~/kanti\_sem\_3/207/Assgn\_3/HTTP\_CLIENT\_SERVER/test\_client$ cat one\_k\_file.txt

start test\_file\_1

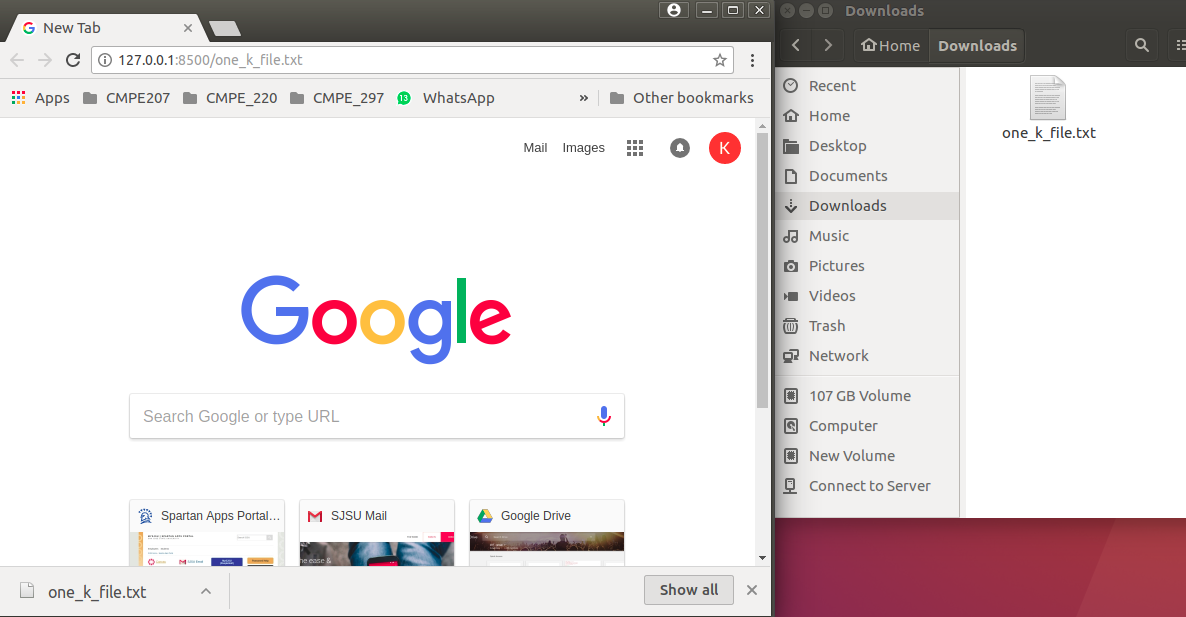
Given a pathname for a file, open() returns a file descriptor, a small, non-negative integer for use in subsequent system calls (read(2), write(2), lseek(2), fcntl(2), etc.). The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

The new file descriptor is set to remain open across an execve(2) (i.e., the FD\_CLOEXEC file descriptor flag described in fcntl(2) is initially disabled). The file offset is set to the beginning of the file (see lseek(2)).

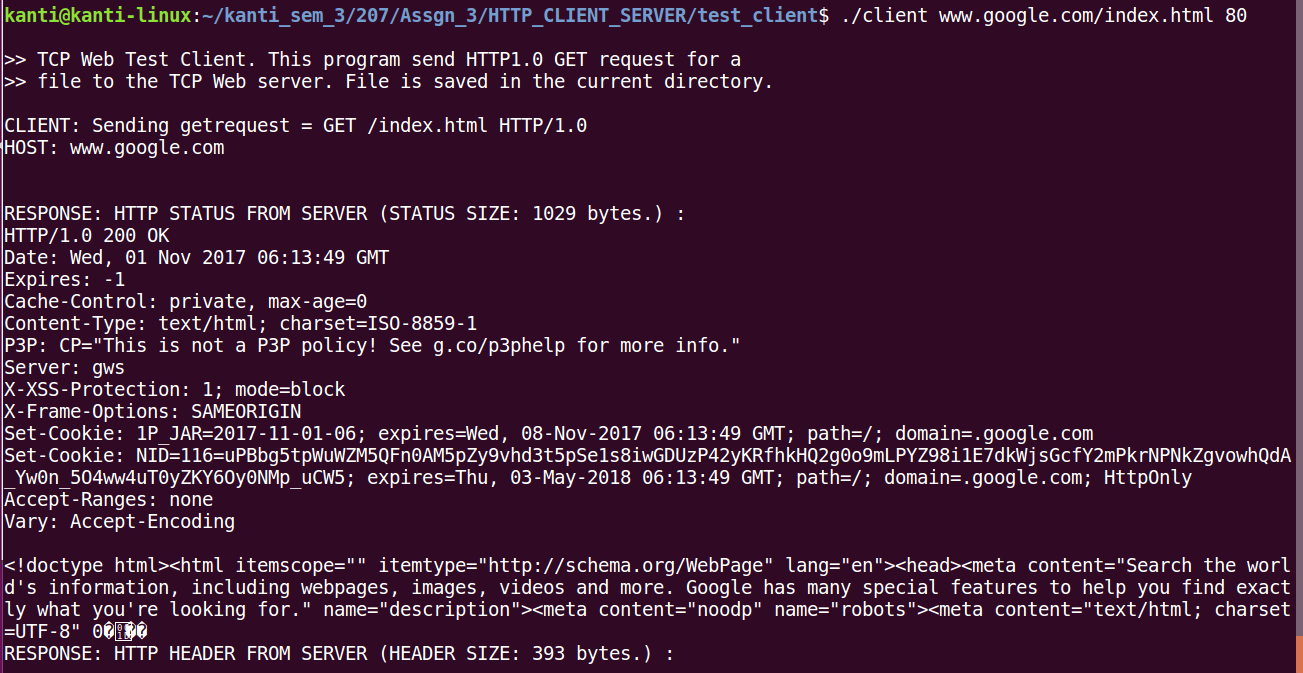
A call to open() creates a new open file description, an entry in the system-wide table of open files. This entry records the file offset and the file status flags (modifiable via the fcntl() F\_SETFL operation). A file descriptor is a reference to one of these entries; this reference is unaffected if pathname is subsequently removed or modified to refer to a different file. The new open file description is initially not shared with any other process, but sharing may arise kanti@kanti-linux:~/kanti\_sem\_3/207/Assgn\_3/HTTP\_CLIENT\_SERVER/test\_client$

**Standard Web Browser connecting to Our server running at port 8500:**

**Output from Web Browser:**



**Our Web Client connecting to Google Web Server on port 80:**



**Reference:**

1. Internetworking with TCP/IP Vol. 3, Client-Server programming and applications, Comer and Stevens, Linux/POSIX Sockets version, ISBN: 0-13-032071-4, 2001.
2. http://www.theinsanetechie.in/2014/02/a-simple-http-client-and-server-in-c.html