**CMPE 207 Lab Assignment 3**

**Team 1: Omkar Rege, Bing Shi, Lam Tran**

Table of Contents

**Server Code2**

**Client Code8**

**Read Me12**

**Test Scenarios13**

**Files for Testing17**

/\***Server Code** \*/

//

// Created by lam on 4/6/17.

//

#include <sys/types.h>

#include <sys/signal.h>

#include <sys/socket.h>

#include <sys/time.h>

#include <sys/resource.h>

#include <sys/wait.h>

#include <sys/errno.h>

#include <netinet/in.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <errno.h>

#include <netdb.h>

#include <stdarg.h>

#include <arpa/inet.h>

#define QLEN 5 /\* maximum connection queue length \*/

u\_short portbase = 0; /\* port base, for non-root servers \*/

void reaper(int);

int processRequest(int fd);

int errexit(const char \*format, ...);

int passivesock(const char \*service, const char \*transport, int qlen);

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

char \*service = "1200"; /\* service name or port number \*/

struct sockaddr\_in fsin; /\* the address of a client \*/

int alen; /\* length of client's address \*/

int msock; /\* master server socket \*/

int ssock; /\* slave server socket \*/

msock = passivesock(service, "tcp", QLEN);

(void) signal(SIGCHLD, reaper);

while (1) {

alen = sizeof(fsin);

ssock = accept(msock, (struct sockaddr \*) &fsin, &alen);

if (ssock < 0) {

if (errno == EINTR)

continue;

errexit("ACCEPT: %s\n", strerror(errno));

}

switch (fork()) {

case 0: /\* child \*/

(void) close(msock);

exit(processRequest(ssock));

default: /\* parent \*/

(void) close(ssock);

break;

case -1:

errexit("fork: %s\n", strerror(errno));

}

}

}

int processRequest(int fd) {

char buf[BUFSIZ];

char fileBuf[BUFSIZ];

char \*reqHeader, \*httpVer, \*requestFile;

int n;

int count = 0;

char const \*filepath = "files";

FILE \*file;

memset((void \*) buf, (int) '\0', BUFSIZ);

while (recv(fd, buf, BUFSIZ, 0) < 0) {

printf("Fail to read from Socket");

}

printf(" ----- %s \n", buf);

fflush(stdout);

reqHeader = strtok(buf, " \n");

if (strncmp(reqHeader, "GET\0", 4) == 0) {

requestFile = strtok(NULL, " \t");

httpVer = strtok(NULL, " \t\n");

printf("ver = %s \n %s", requestFile, httpVer);

if (strncmp(httpVer, "HTTP/1.0", 8) != 0 && strncmp(httpVer, "HTTP/1.1", 8) != 0) {

send(fd, "HTTP/1.0 400 Bad Request\n", 25, 0);

printf("HTTP/1.0 400 Bad Request");

}

char fullpath[sizeof(filepath) + sizeof(requestFile) + 1];

strcpy(fullpath, filepath);

strcat(fullpath, requestFile);

strcat(fullpath, "\0");

file = fopen(fullpath, "r");

n = 0;

if (file) {

send(fd, "HTTP/1.0 200 OK\n\n", 17, 0);

while ((n = fread(fileBuf, 1, sizeof(fileBuf) - 1, file)) > 0) {

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

fflush(stdout);

send(fd, fileBuf, n, 0);

}

fclose(file);

close(fd);

} else {

printf("File %s does not exis\n", fullpath);

send(fd, "HTTP/1.0 404 Not Found\n", 23, 0);

printf("HTTP/1.0 404 Not Found\n");

}

close(fd);

return 0;

}

}

int passivesock(const char \*service, const char \*transport, int qlen) {

struct servent \*pse; /\* pointer to service information entry \*/

struct protoent \*ppe; /\* pointer to protocol information entry\*/

struct sockaddr\_in sin; /\* an Internet endpoint address \*/

int s, type; /\* socket descriptor and socket type \*/

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

sin.sin\_family = AF\_INET;

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

/\* Map service name to port number \*/

if (pse = getservbyname(service, transport))

sin.sin\_port = htons(ntohs((u\_short) pse->s\_port) + portbase);

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

errexit("can't get \"%s\" service entry\n", service);

if ((ppe = getprotobyname(transport)) == 0)

errexit("can't get \"%s\" protocol entry\n", transport);

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

type = SOCK\_DGRAM;

else

type = SOCK\_STREAM;

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

if (s < 0)

errexit("can't create socket: %s\n", strerror(errno));

int reuse = 1;

/\*if (setsockopt(s, SOL\_SOCKET, SO\_REUSEADDR, (const char \*) &reuse, sizeof(reuse)) < 0)

perror("setsockopt(SO\_REUSEADDR) failed");

if (setsockopt(s, SOL\_SOCKET, SO\_REUSEPORT, (const char \*) &reuse, sizeof(reuse)) < 0)

perror("setsockopt(SO\_REUSEPORT) failed");\*/

if (bind(s, (struct sockaddr \*) &sin, sizeof(sin)) < 0)

errexit("can't bind to %s port: %s\n", service,

strerror(errno));

if (type == SOCK\_STREAM && listen(s, qlen) < 0)

errexit("can't listen on %s port: %s\n", service,

strerror(errno));

return s;

}

/\*--------------------------------------------------HELPER--------------------------------------------------\*/

void reaper(int sig) {

int status;

while (wait3(&status, WNOHANG, (struct rusage \*) 0) >= 0)

/\* empty \*/;

}

int errexit(const char \*format, ...) {

va\_list args;

va\_start(args, format);

vfprintf(stderr, format, args);

va\_end(args);

exit(1);

}

/\***Client Code** \*/

//

// Created by lam on 4/6/17.

//

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <stdarg.h>

#include <errno.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <ctype.h>

#include <time.h>

#define LINELEN 4096

#ifndef INADDR\_NONE

#define INADDR\_NONE 0xffffffff

#endif /\* INADDR\_NONE \*/

int requestFile(const char \*host, const char \*service, const char \*filename);

int errexit(const char \*format, ...);

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

const char \*transport);

int parseToken(char\* header);

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

char \*fileName = NULL;

char \*host = "localhost"; /\* host to use if none supplied \*/

char \*service = "1200"; /\* default service name "8081" \*/

switch (argc) {

case 2:

fileName = argv[1];

break;

case 3:

host = argv[1];

fileName = argv[2];

break;

default:

fprintf(stderr, "usage: TCPCLient [filename]\n");

exit(1);

}

clock\_t start = clock();

requestFile(host, service, fileName);

clock\_t end = clock();

double time\_diff = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Time Required to Fetch the file is %f",time\_diff); fflush(stdout);

exit(0);

}

int requestFile(const char \*host, const char \*service, const char \*filename) {

char buf[LINELEN + 1]; /\* buffer for one line of text \*/

int s, n; /\* socket descriptor, read count\*/

char \*header = malloc(sizeof(filename) + sizeof(char) \* 17);

sprintf(header, "GET /%s HTTP/1.0\n\n", filename);

FILE \*file;

printf("--- %s", header);

fflush(stdout);

s = connectsock(host, service, "tcp");

int count = send(s, header, strlen(header), 0);

printf("--- %d \n", strlen(header));

file = fopen(filename, "w+");

int isHeader=1; char\* token = NULL; char\* resp2 = NULL;

while ((n = recv(s, buf, sizeof(buf), 0)) > 0) {

if(isHeader==1) {

fputs(buf, stdout);

resp2 = strstr(buf,"\n\n");

if(resp2==NULL){ resp2 = strstr(buf,"\r\n\r\n");}

isHeader=0;

fputs(resp2+2, file);

memset(buf, 0, sizeof(buf));

continue;

}

if(isHeader==0)

fputs(buf, file);

memset(buf, 0, sizeof(buf));

}

fclose(file);

free(header);

}

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

struct hostent \*phe; /\* pointer to host information entry \*/

struct servent \*pse; /\* pointer to service information entry \*/

struct protoent \*ppe; /\* pointer to protocol information entry\*/

struct sockaddr\_in sin; /\* an Internet endpoint address \*/

int s, type; /\* socket descriptor and socket type \*/

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

sin.sin\_family = AF\_INET;

/\* Map service name to port number \*/

if (pse = getservbyname(service, transport))

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

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

errexit("can't get \"%s\" service entry\n", service);

/\* Map host name to IP address, allowing for dotted decimal \*/

if (phe = gethostbyname(host))

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

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

errexit("can't get \"%s\" host entry\n", host);

/\* Map transport protocol name to protocol number \*/

if ((ppe = getprotobyname(transport)) == 0)

errexit("can't get \"%s\" p rotocol entry\n", transport);

/\* Use protocol to choose a socket type \*/

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

type = SOCK\_DGRAM;

else

type = SOCK\_STREAM;

/\* Allocate a socket \*/

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

if (s < 0)

errexit("can't create socket: %s\n", strerror(errno));

/\* Connect the socket \*/

if (connect(s, (struct sockaddr \*) &sin, sizeof(sin)) < 0)

errexit("can't connect to %s.%s: %s\n", host, service,

strerror(errno));

return s;

}

int errexit(const char \*format, ...) {

va\_list args;

va\_start(args, format);

vfprintf(stderr, format, args);

va\_end(args);

exit(1);

}

### Readme

To compile Server :

1. Open terminal at code folder
2. gcc -o Server.o Server.c

3. Create a source dir named “files” and keep respective files in i

To run Server :

./Server.o

To compile Client :

1. Open terminal at code folder

2. gcc -o Client.o Client.c

To run Client :

./Client.o “file\_name” For example if “files” dir at server side contains file1.txt we can run client program like,

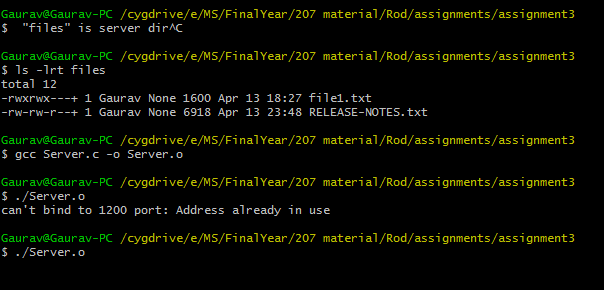
./Client.o “file1.txt”

This command will transfer file1.txt from server to client’s current location.

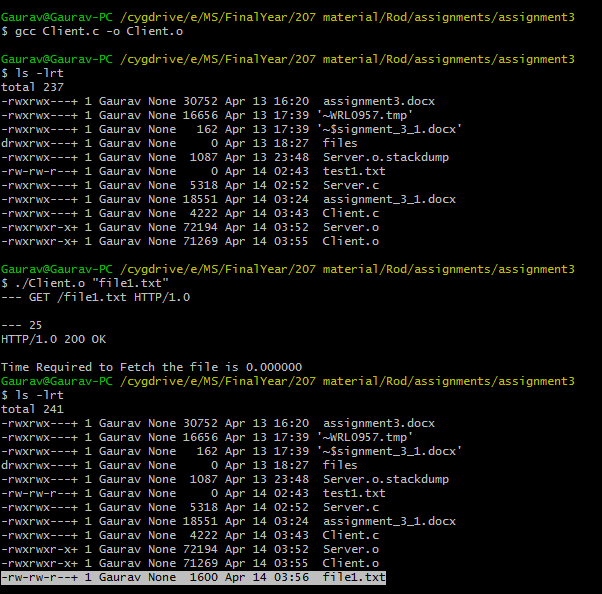
**Testing Scenarios**:

Part A Testing Server

1. Testing our Server with our client.



Now sending request to transfer file1.txt

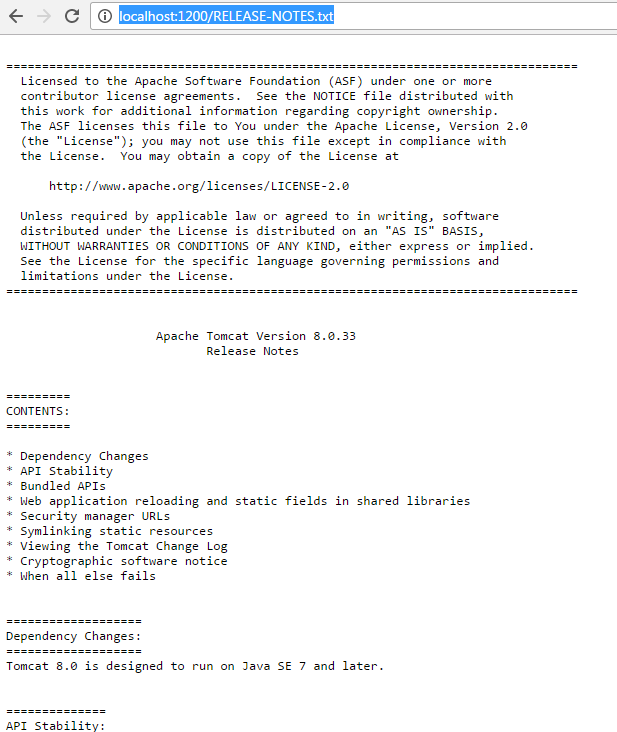


Headers has been parsed with status 200.

1. Testing our server with Browser.

Request url : <http://localhost:1200/RELEASE-NOTES.txt>

(Please replace RELEASE-NOTES.txt with your file name and place it in server’s “files” dir)

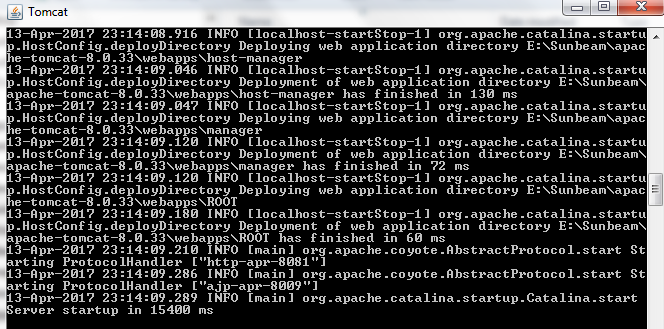


B Testing Client with a standard WebServer

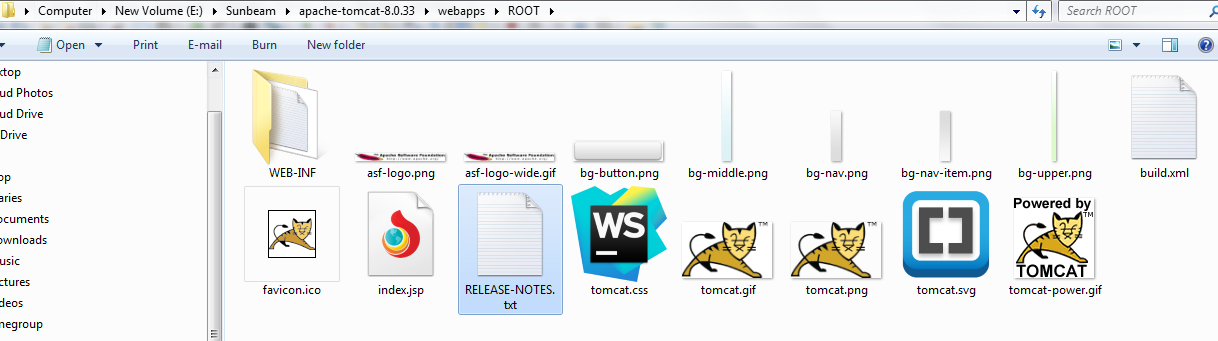
We tested our client with Apache Tomcat webserver and was able to retrieve files.

(Note: Please change port to Tomcat’s assigned port number in Client.c and compile)

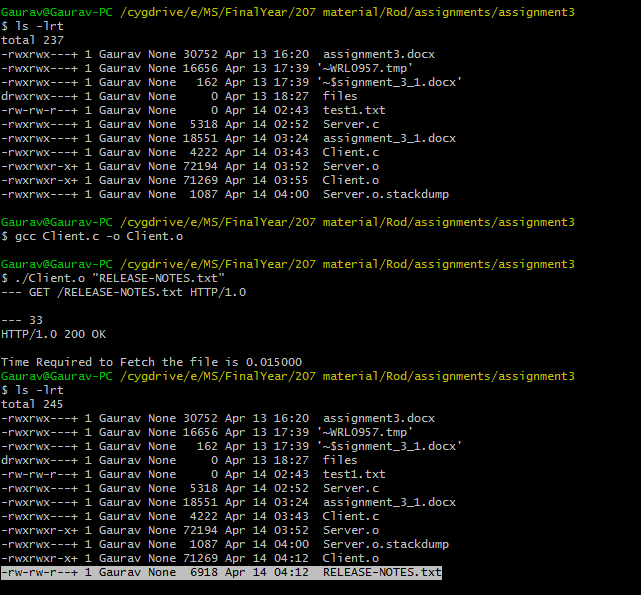




Place a file in Tomcat’s webapp dir,



Now request the request RELEASE\_NOTES.txt in a similar way as described earlier,



Hence we were able to test our client and server with real webserver as well as actual browser parsing/Constructing HTTP headers.

**Files for Testing :**

**file1.txt**

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

**file2.txt**

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa