PROJECT 1 : COMPUTER NETWORKS (CS425A)

HTTP SERVER IMPLEMENTATION

19.08.2016

—

PRAMOD CHUNDURI
ROLL NO 13221
chpramod@iitk.ac.in

IMPLEMENTED OPTIONS :

HYPERLINKED DIRECTORY LISTING
SERVER PORT INITIALIZED AT STARTUP (CLI)
SERVER BASE DIRECTORY INITIALIZED AT STARTUP

Overview of Optional Features

I. Directory Listing

The list of directories and files in the requested directory is provided to the client, if the URI requested does not have an “index.html” file.

II. Port Number and Base directory on startup

Port Number and Base directory are initialized on startup. So, the server expects a command of the format

$./server port_no base_dir

Additional feature : 404 Error Response

In case the requested file is not found, the server throws a 404 exception, with an explicit error message.

Testing Results

I. TESTING METHODS

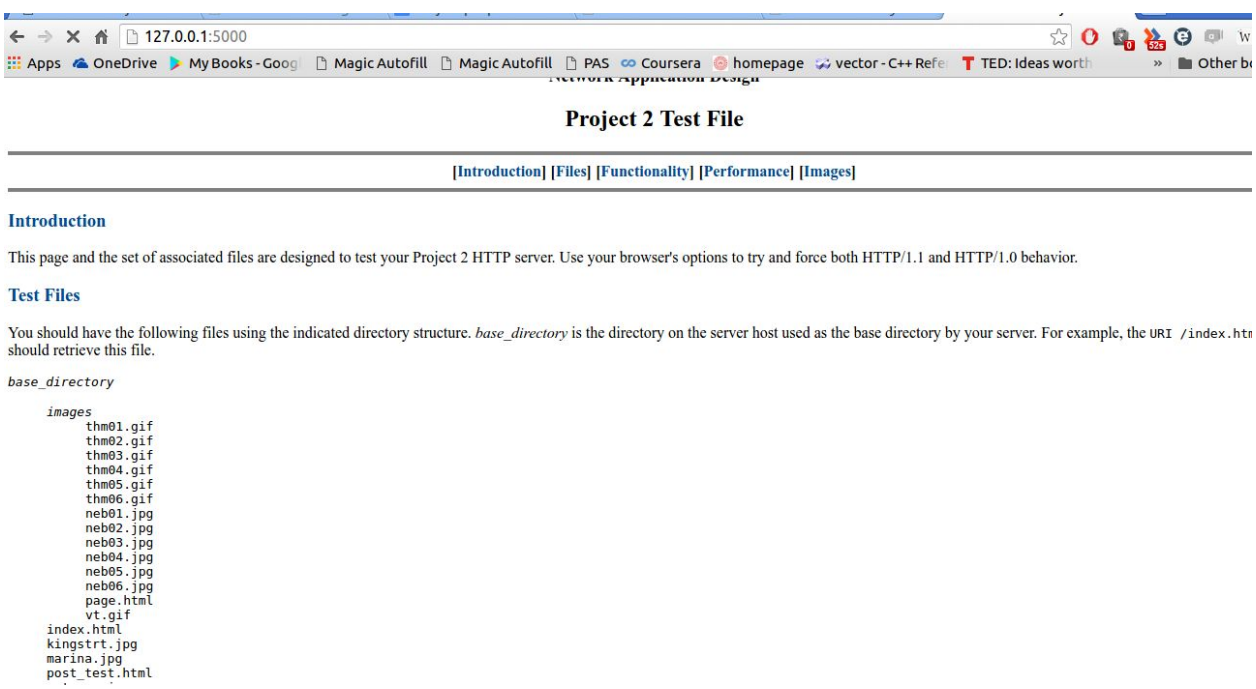
- All the implemented features have been tested either analyzing the response on terminal or verifying browser output.
-
- For testing, chrome and firefox browsers have been used.

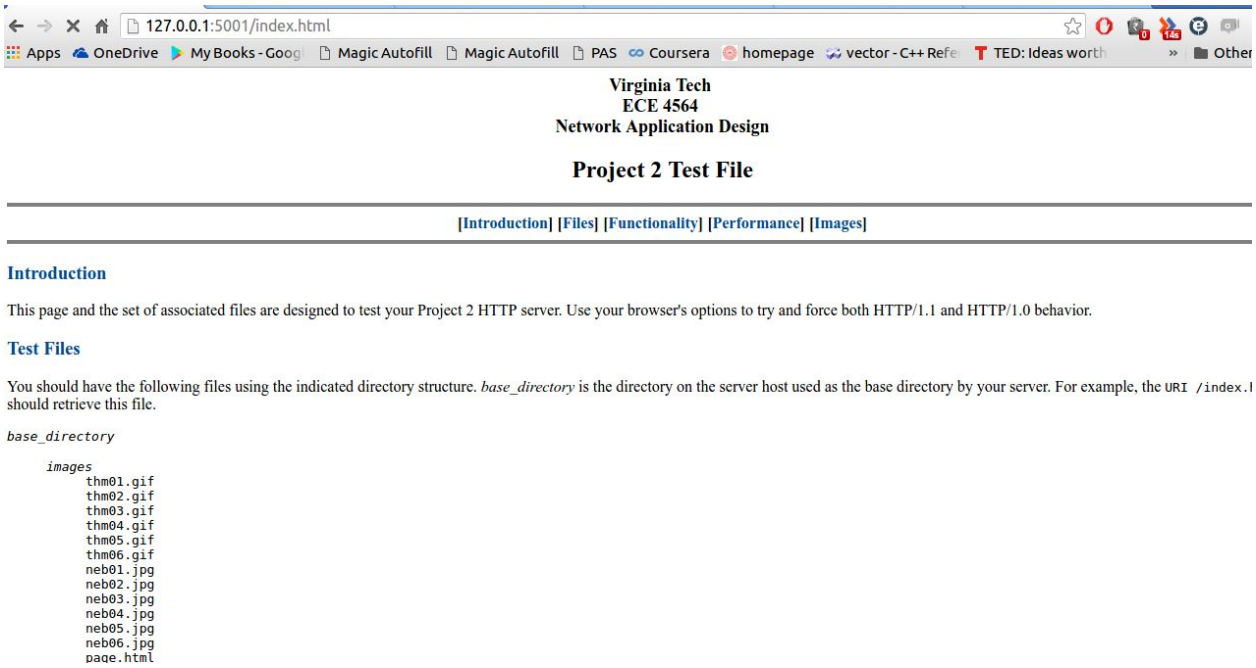
II. SCREENSHOTS

Server execution format

```
pramod@pramod:Project 1$ ./server 5001 webfiles
Successfully served a client
```

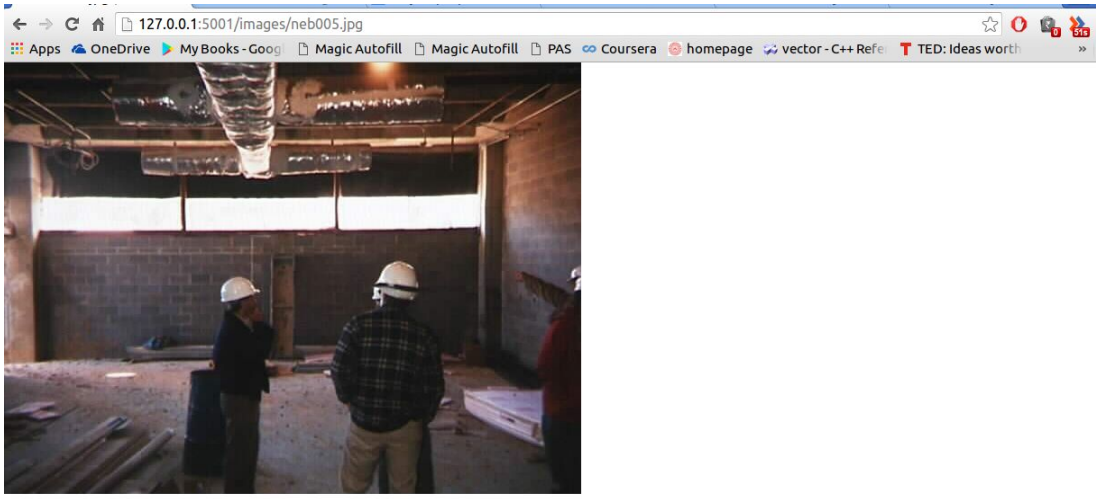
Successful response for / (with and without index.html)



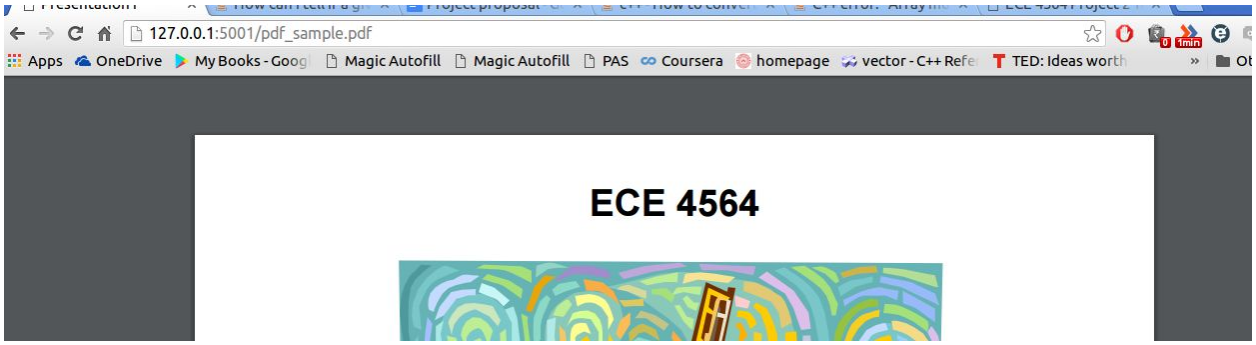


Another Page test case

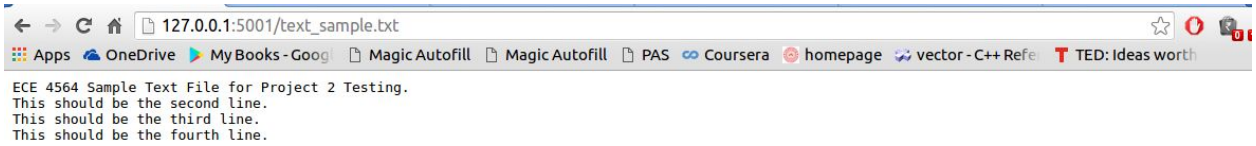
Image test case



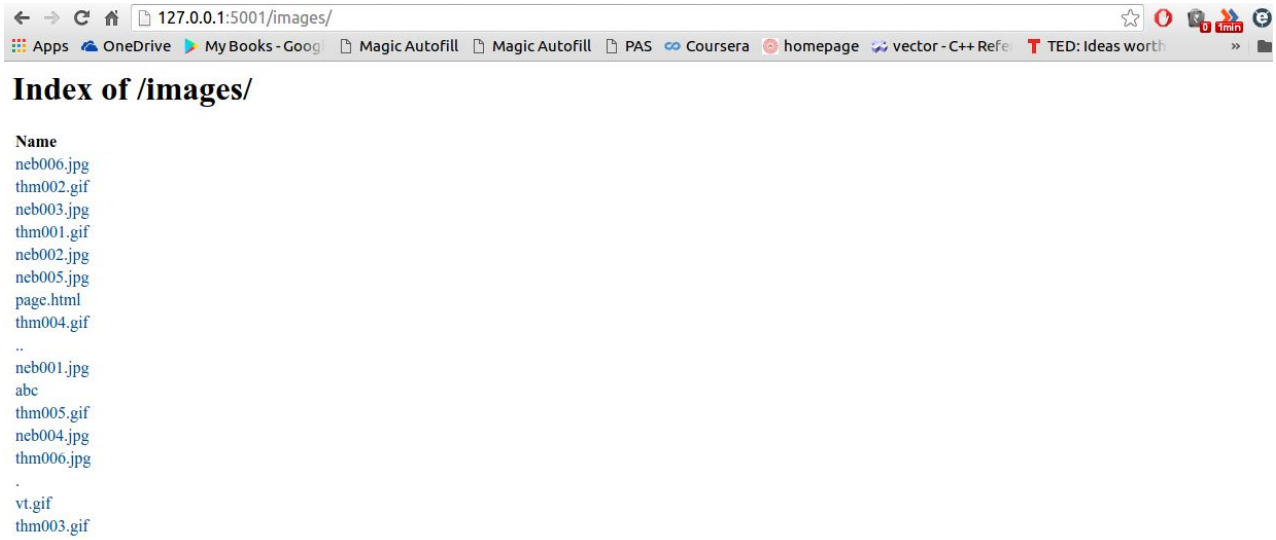
PDF test case



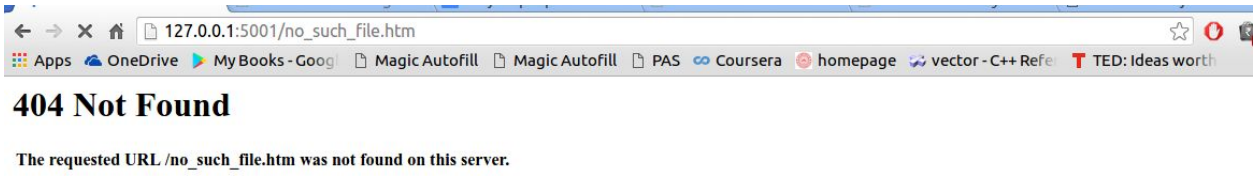
TXT test case



Directory Listing



File Not Found error



Summary

All the implemented features are working fine as seen from the screenshots.

Appendix (Code)

```
#include <bits/stdc++.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <string>
#include <dirent.h>
#include <sys/stat.h>

#define SIZE 2048
using namespace std;
//Structure to store the GET request entities
struct parsed_request{
	string method;
	string path;
	string protocol;
	string version;
	string host;
	string connection;
};
string directory;
//Function to serve the client
int serve_the_client(int arg){
	while (1){
```

```
int newsockid = arg;
char *endIndex;
char request[SIZE];
char response_header[2048];
char response_message[4096];
string port;
int bytes_left = sizeof(request);
char* curr_ptr = request;
int curr = 0;
int recv_curr = 0;
bzero(request,SIZE);
map<string,string> name;
//do-while loop to receive GET request
do{
	recv_curr = recv(newsockid,curr_ptr,SIZE,0);
	endIndex = strstr(request, "\r\n\r\n");
	if (recv_curr == 0){
		break;
	}
	curr_ptr = curr_ptr + recv_curr;
	if (recv_curr <= 0){
		printf("Error in connection.\n");
		return -1;
	}
}while(endIndex==NULL);
//Preprocessing to parse the GET request
parsed_request curr_req;
vector<char*> lines;
char* saveptr;
```

```
char* currPtr = strtok_r(request,"\r\n",&saveptr);
while (currPtr!=NULL){
	lines.push_back(currPtr);
	currPtr = strtok_r(NULL,"\r\n",&saveptr);
}
//vector lines stores individual lines of the GET request
for (int i=0;i<lines.size();i++){
	if (i == 0){
		//i =0 is the case of Request line
		stringstream ss(lines[0]);
		int count = 0;
		string temp_first;
		//iterating to get method, path and version
		while (getline(ss,temp_first,' ')){
			if (count ==0){
				curr_req.method=temp_first;
				fflush(stdout);
			}
			else if (count ==1){
				curr_req.path = temp_first;
			}
			else if (count == 2){
				curr_req.version = temp_first;
			}
			count++;
		}
	}
	else{
		// all the lines other than request line
```

```
                    stringstream ss(lines[i]);
                    string temp_next;
                    int count = 0;
                    string left,right;
                    //iterating and mapping each option with the value, Eg: For
the line "Host: xyz.com:2000", "Host" is mapped with "xyz.com:2000"
                    while (getline(ss,temp_next,':')){
                        if (count == 0){
                            left = temp_next;
                        }
                        else if (count == 1){
                            temp_next.erase(0,1);
                            right = temp_next;
                            name[left]=right;
                            if (left != "Host"){
                                count = 0;
                                continue;
                            }
                        }
                        else if (count == 2){
                            port = temp_next;

                        }
                        count++;
                    }
                }
            }
            curr_req.host = name["Host"];
            curr_req.connection = name["Connection"];
            // string directory = "webfiles";
```

```
string full_path = directory + curr_req.path;
string file_req = full_path.substr(full_path.find_last_of("/")+1);
string new_full_path;
struct stat s;
//Got the necessary GET request information
if( stat(full_path.c_str(),&s) == 0 ){
        //Checking whether requested path is file or a directory, the library
'sys/stat.h' has been used
        //reference :
http://stackoverflow.com/questions/146924/how-can-i-tell-if-a-given-path-is-a-directory-o
r-a-file-c-c
    if( s.st_mode & S_IFDIR ){
        //if it is a directory
      int temp_flag;
      //checking whether "/" is given in the client's request or not
if (file_req == ""){
                 new_full_path = full_path + "index.html";
                temp_flag = 0;
            }
            else{
                new_full_path = full_path + "/index.html";
                temp_flag = 1;
            }
            //Checking whether index.html is present in the requested
path
            FILE *file_to_send =fopen(new_full_path.c_str(), "rb");
            if (file_to_send == NULL){
                //Code for Directory Listing, run when index.html is
not found
                char html_message[4096];
                char html_header[4096];
```

```
string full_host = curr_req.host + ":" + port;
//html code is generated by the server and sent to the client
sprintf(html_message,"<html><head>"
        "<title>Index of %s </title>"
        "</head>"
        "<body>"
        "<h1>Index of %s</h1>"
        "<table>"
"<tr><td><strong>Name</strong><td></tr>",curr_req.path.c_str(), curr_req.path.c_str());
//DIRENT library used for listing the directories and files
//reference used : http://stackoverflow.com/questions/4204666/how-to-list-files-in-a-directory-in-a-c-program
DIR *mydir;
struct dirent *dir;
mydir = opendir(full_path.c_str());
if (mydir){
    while ((dir = readdir(mydir)) != NULL){
        string full_dir_addr;
        if (temp_flag ==0){
            full_dir_addr = full_path + dir->d_name;
        }
        else{
            full_dir_addr = full_path + "/"+ dir->d_name;
        }
        if (stat(full_dir_addr.c_str(),&s) == 0){
```

```
                                                    string temp_ref;
                                                    if (temp_flag ==0){
                                                            temp_ref =curr_req.path +
dir->d_name;
                                                    }
                                                    else{
                                                            temp_ref =curr_req.path +
"/"+dir->d_name;
                                                    }

sprintf(html_message+strlen(html_message),"<tr><td><a href='%s'>
%s</a><td></tr>",temp_ref.c_str(),dir->d_name);
                                                    }
                                            }
                                            closedir(mydir);
                                    }

sprintf(html_message+strlen(html_message),"</table></body></html>");
                                    //html content for directory listing generated, this is
now sent to the server along with the header
                                    int cLength = strlen(html_message);
                                    string cType = "text/html";
                                    sprintf(html_header,"%s 200
OK\r\n",curr_req.version.c_str());

sprintf(html_header+strlen(html_header),"Content-Length: %d\r\n",cLength);

sprintf(html_header+strlen(html_header),"Content-Type: %s\r\n\r\n",cType.c_str());
                                    char *curr_ptr = html_header;
                                    int curr = 0;
                                    int bytes_left = strlen(html_header);
                                    while (bytes_left >0){
```

```
                        curr = send(newsockid,curr_ptr,bytes_left,0);
                        bytes_left = bytes_left - curr;
                        curr_ptr = curr_ptr + curr;
                    }
                    curr_ptr = html_message;
                    curr = 0;
                    bytes_left = strlen(html_message);
                    while (bytes_left >0){
                        curr = send(newsockid,curr_ptr,bytes_left,0);
                        bytes_left = bytes_left - curr;
                        curr_ptr = curr_ptr + curr;
                    }
                    return 0;
                }
                fclose(file_to_send);
            }
            else if( s.st_mode & S_IFREG )
            {
                //if it is a file
                new_full_path = full_path;
            }
            else
            {
                //if it is neither a file nor a directory
                printf("invalid object\n");
                return -1;
            }
        }
        FILE *file_to_send =fopen(new_full_path.c_str(), "rb");
```

```
if (file_to_send != NULL){
        //Case when file is found, below code to serve the file
        fseek(file_to_send,0,SEEK_END);
        int cLength = ftell(file_to_send);
        rewind(file_to_send);
        string ext = full_path.substr(full_path.find_last_of(".")+1);
        string cType;
        //check for requested file's extension
        if (ext == "html" || ext == "htm"){
                cType = "text/html";
        }
        else if (ext == "txt"){
                cType = "text/plain";
        }
        else if (ext == "jpeg" || ext == "jpg"){
                cType = "image/jpeg";
        }
        else if (ext == "gif"){
                cType = "image/gif";
        }
        else if (ext == "pdf"){
                cType = "Application/pdf";
        }
        else{
                cType = "unknown";
        }
        //response header and message created for requested file
        sprintf(response_header,"%s 200 OK\r\n",curr_req.version.c_str());
        sprintf(response_header+strlen(response_header),"Content-Length:
%d\r\n",cLength);
```

```
sprintf(response_header+strlen(response_header),"Content-Type: %s\r\n\r\n",cType.c_str());
char *curr_ptr = response_header;
int curr = 0;
int bytes_left = strlen(response_header);
while (bytes_left >0){
	curr = send(newsockid,curr_ptr,bytes_left,0);
	bytes_left = bytes_left - curr;
	curr_ptr = curr_ptr + curr;
}
while (!feof(file_to_send)){
	bzero(response_message,sizeof(response_message));
	int bytes_read = fread(response_message,1,sizeof(response_message),file_to_send);
	if (bytes_read == 0){
		break;
	}
	curr_ptr = response_message;
	curr = 0;
	bytes_left = bytes_read;
	while (bytes_left >0){
		curr = send(newsockid,curr_ptr,bytes_left,0);
		bytes_left = bytes_left - curr;
		curr_ptr = curr_ptr + curr;
	}
}
//Checks for persistency of connection
if (curr_req.connection == "close"){
	return 1;
}
```

```
                if (recv_curr == 0){
                        return 2;
                }
        }
        else{
                //Case when the requested file is not found
                sprintf(response_header,"%s 404 Not
Found\r\n",curr_req.version.c_str());
                sprintf(response_header+strlen(response_header),"Content-Type:
%s\r\n\r\n","text/html");
                curr_ptr = response_header;
                curr = 0;
                bytes_left = strlen(response_header);
                while (bytes_left >0){
                        curr = send(newsockid,curr_ptr,bytes_left,0);
                        bytes_left = bytes_left - curr;
                        curr_ptr = curr_ptr + curr;
                }
                //HTML content created with error message and sent to the client
                char error_message[1024];
                sprintf(error_message,"<html><head>"
                                "<title>404 Not Found </title>"
                                "</head>"
                                "<body>"
                                "<h1>404 Not Found</h1>"
                                "<table>"
                                "<tr><td><strong>The requested URL %s
was not found on this server.</strong><td></tr>",curr_req.path.c_str());
                curr_ptr = error_message;
                curr = 0;
```

```
                        bytes_left = strlen(error_message);
                        while (bytes_left >0){
                                curr = send(newsockid,curr_ptr,bytes_left,0);
                                bytes_left = bytes_left - curr;
                                curr_ptr = curr_ptr + curr;
                        }
                        return 0;
                }
                fclose(file_to_send);
        }
        return 0;

}

int main(int argc, char *argv[]){
        int numThreads = 0;
        if (argc != 3){
                printf("The program expects two arguments, port number, base
directory\n");
        }
        else{
                //socket creation
                int port_no = atoi(argv[1]);
                directory = argv[2];
                // directory = string(base_directory);
                int sockid = socket(AF_INET,SOCK_STREAM,0);
                struct sockaddr_in addr;
                struct sockaddr clientAddr;
                //defining socket properties
                socklen_t addrLen;
```

```
addr.sin_family = AF_INET;
addr.sin_port = htons(port_no);
addr.sin_addr.s_addr = htonl(INADDR_ANY);
//binding the socket to the network interface
int bind_status = bind(sockid,(struct sockaddr *)&addr,sizeof(addr));
if (bind_status == 0){
	int listen_status = listen(sockid, SOMAXCONN);
	if (listen_status == 0){
		while (1){
			//Listen for client requests
			int newsockid = accept(sockid, &clientAddr,&addrLen);
			//Forking a child process for every accepted
connection
			//Parent process runs in an infinite loop until
termination
			pid_t pid = fork();
			numThreads++;
			if (pid == 0){
				//Call to the function serving the client
				int status = serve_the_client(newsockid);
				//Appropriate messages on the command line,
for different exit statuses
				if (status == -1){
					printf("An error occured\n");
				}
				if (status == 0){
					printf("Successfully served a client\n");
				}
				if (status == 1){
```

```
                                        printf("Request to close connection from
the client\n");
                                }
                                if (status == 2){
                                        printf("Connection closed by the
client\n");
                                }
                                fflush(stdout);
                                exit(0);
                        }
                        else if (pid > 0){
                                //parent process continues in while loop
indefinitely, until Ctrl + C is found
                                continue;
                        }
                        else
                {
                        // fork was not successful
                        printf("fork() failed!\n");
                        return 1;
                }
                }
                close(sockid);
        }
}
else{
        char errMsg[128] = "Could not bind to the specified port\n\0";
        fwrite(errMsg,1,strlen(errMsg),stderr);
        close(sockid);
}
```

```
        }
        return 0;
}
```