```
//ICSI 333. System Fundamentals
//Peter Buonaiuto: 001497626
//TA Omkar Kulkarni
//Spring 2022

//Program to host a web server given a directory of files, and allow a client to connect using telnet or nc to request files from the server in the form /GET /file.txt HTTP/1.0

/*

Work division: Project Partner: Julian
    Most work was done while communicating:
        -Collabbed on plans, logic, execution
        //------TEMPLATE---//
        -check args
        -client
            -send requests and read processed requests
            -both are infinite loops
            -both have read checks for if done or continuing
        -server
            -read requests from client
                -use 3D array; list of lists of 3 args
            -process requests
                -loop for number of requests and create thread for each
            -send processed request to client
                -have to put everyting together so prints all of each thread at once


    Specific Individual pieces created by Peter
        -3D array
        -thread struct
    Specific Individual pieces created by Julian
        -directory alteration and check
        -file processing
```

```
*/

//Will listen for requests until user types done constant.
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <pthread.h>

//Global constants
#define PORT 8001 //server port to host
#define SA struct sockaddr //server socket typedef
#define MAX_ARGS 3 //arguments for a client request
#define EXIT_MSG "done\n" //will finish requesting
#define MAX_REQUESTS 10 //max number of files a client can request in one session
#define ARG_ERR_MSG "Usage: GET [file] [html version]\nUsage: done\n"

char args[MAX_REQUESTS][MAX_ARGS][256];  //Arguments of requests
int sockfd, client_fd, requestCount; //Sockets and number of client requests

//Create a thread structure to pass through to the thread function
typedef struct _thread_data_t
{
```

```
  int tid; //Request ID number (thread #)
  char args[MAX_ARGS][256]; //request arguments, split up
} thread_data_t;

//Server functions
void quit(int);
void processRequests();
void *thr_func(void *);
void setDir(char *);
void startServer();
void serverExec(int);

//Main checks if we're properly hosting the server with directory argument
int main(int argc, char *argv[])
{
	if(argc !=2)
	{
		printf("Usage: ./webserver [DIRECTORY]\n");
		return -1;
	}
	//Set directory, or exit with failure.
	setDir(argv[1]);
	startServer();
}

//Start the server on the given directory and listen for connections
void startServer()
{
	//Bind SIGINT to my quit function to safely close socket
	signal(SIGTSTP, quit);

	unsigned int len;
	struct sockaddr_in servaddr, cli;

		// socket creation
```

```
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd == -1) {
                printf("socket creation failed...\n");
                exit(0);
        }
        bzero(&servaddr, sizeof(servaddr));

// assign
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(PORT);

// Binding
if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
        exit(0);
}
//listen for client
if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
}

struct in_addr ipAddr = servaddr.sin_addr;
char str_ip[INET_ADDRSTRLEN];
inet_ntop( AF_INET, &ipAddr, str_ip, INET_ADDRSTRLEN );

printf("Server listening at %s on %d.. \n",str_ip, PORT);
len = sizeof(cli);

// Accept the client
client_fd = accept(sockfd, (SA*)&cli, &len);
if (client_fd < 0)
{
        printf("server accept failed...\n");
```

```
            exit(0);
        }
        else printf("Client accepted!\n");
        //Server's execution chain
        serverExec(client_fd);
        // After requests, close socket
        close(sockfd);
}

//Server will execute once a client connects
void serverExec(int client_fd)
{
        do
        {
            char response[256];   //The text supplied by the user
            memset ( response, '\0', sizeof(response));
            read(client_fd, response, sizeof(response));

            //We stored the response, now extract the arguments
            char * token = strtok(response, " ");
            int argi = 0;

            //Store each argument of the given command
        while(token != NULL)
        {
            //Put the valid argument into our argument array
            strcpy(args[requestCount][argi++], token);
            //Avoid seg fault if too many arguments
            if (argi > MAX_ARGS)
                break;
            //The following code avoids including the newline from ENTER in the token
            if (argi == MAX_ARGS - 1)
                token = strtok(NULL, "\n");
            else
            token = strtok(NULL, " ");
```

```
        }
        //Are there not 3 arguments?
        if (argi != MAX_ARGS)
        {
                //Are we trying to quit?
                if (argi == 1)
                {
                        //Yes, done
                        if (strcmp(args[requestCount][0], EXIT_MSG) == 0)
                        {
                                processRequests();
                                break; //Stops infinite loop and stops server
                        }
                        else
                        {
                                //We entered one argument and it wasn't exit. Display usage
                                write(client_fd, ARG_ERR_MSG, strlen(ARG_ERR_MSG));
                                continue;
                        }
                }
                //We entered 2 which is always bad
                write(client_fd, ARG_ERR_MSG, strlen(ARG_ERR_MSG));
                continue;
        }
        //Here all args are split, and there are 3. Check if the command is valid
        if (strcmp(args[requestCount][0], "GET") != 0)
        {
                write(client_fd, ARG_ERR_MSG, strlen(ARG_ERR_MSG));
                continue;
        }

    //  Here we have all valid arguments for the request.
    //      for(int i = 0; i< argi; i++)
//          printf("Argument %d: %s\n",i+1, args[requestCount][i]);
```

```
            requestCount++;
        } while (1);
}

//Process each request by making a thread for each
void processRequests()
{
        pthread_t thr[requestCount];
    thread_data_t thr_data[requestCount];

    //make threads
    for (int i = 0; i < requestCount; ++i)
    { //Save thread (request) ID
        thr_data[i].tid = i;

            // put this request's arguments in the thread structure
            for(int j = 0; j < MAX_ARGS; j++)
            {
                //printf("THREAD %d, ARG #%d: %s\n", i, j, args[i][j]);
                strcpy(thr_data[i].args[j], args[i][j]);
            }
            //Make the thread with the complete structure
        pthread_create(&thr[i], NULL, thr_func, &thr_data[i]);
    }

    //block until threads complete
     for (int i = 0; i < requestCount; ++i)
            pthread_join(thr[i], NULL);
} //finished processing each request

//Thread Function (for each request)
void *thr_func(void *arg)
{
        int fd; //Descriptor for current file.
```

```
    //Cast the argument (from creation) to store req arguments
thread_data_t *data = (thread_data_t *)arg;
    //write this to client
    char label[BUFSIZ];
    bzero(label, sizeof(label));
    sprintf(label, "\nRequest Thread #%d: (%s)\n", data->tid, data->args[1]);

    //get current dir
    char buff[BUFSIZ];
    bzero(buff, sizeof(buff));
    getcwd(buff, sizeof(buff));
    char temp[BUFSIZ];
    strcpy(temp, buff);
    //append file to current dir
    strcat(temp, data->args[1]);
    //printf("file path: %s\n", temp);fflush(stdout);

    //open file, read, close
    if((fd = open(temp, O_RDONLY)) == -1){// open file at temp
            //write to client
            char error[] = "Error 404\n";
            strcat(label, error);
            write(client_fd, label, sizeof(label));

    }
    else{
            //send to client
            char buf[BUFSIZ];
            strcpy(buf, label);

            //append html version and result
            strcat(buf, data->args[2]);
            strcat(buf, " 200 OK\n");

            char temp[BUFSIZ];
```

```
            read(fd, temp, sizeof(temp));// read file into buf

            char contL[BUFSIZ];
            bzero(contL, sizeof(contL));
            sprintf(contL, "Content-Length: %zu\n\n", strlen(temp));
            strcat(buf, contL);
            strcat(buf, temp);
            strcat(buf, "\n");
            write(client_fd, buf, sizeof(buf));
            close(fd);
        }

    pthread_exit(NULL);
}

void setDir(char *in){
//Make sure its a directory
        struct stat pathdes;
        stat(in, &pathdes);
        if(S_ISDIR(pathdes.st_mode) != 1){
            printf("Destination isn't a dir\n");fflush(stdout);
            quit(sockfd);
        }

        //change to specified directory
        DIR *dir;

        if((dir = opendir(in)) == NULL)
        {// opening dir dest
            printf("Failure opening directory\n");fflush(stdout);
            quit(sockfd);
        }
        struct dirent *dirp;
        chdir(in);// changing working dir to destination
}
```

```
//Handle quitting, so we close the socket so we avoid future binding errors
void quit(int socket)
{
	close(sockfd);
	exit(1);
}
```