ΕΠΛ371 – Προγραμματισμός Συστημάτων

Εργασία:	04 (Ομαδική)	
Ημ. Ανάθεσης:	8 Απριλίου 2015	
Ημ. Παράδοσης:	5 Μαΐου 2015	
Ονόματα:	Ανδρέου Ανδρέας	Αλεξάνδρου Σάββας
Ταυτότητες:	983759	985920

Στόχος:

Στόχος της 4ης ομαδικής προγραμματιστικής άσκησης για το μάθημα ΕΠΛ371 – Προγραμματισμός Συστημάτων, είναι η ανάπτυξη ενός πολυνηματικού εξυπηρετητή ιστού για το πρωτόκολλο HTTP, οποίος θα υποστηρίζει ένα βασικό υποσύνολο της έκδοσης 1.1.

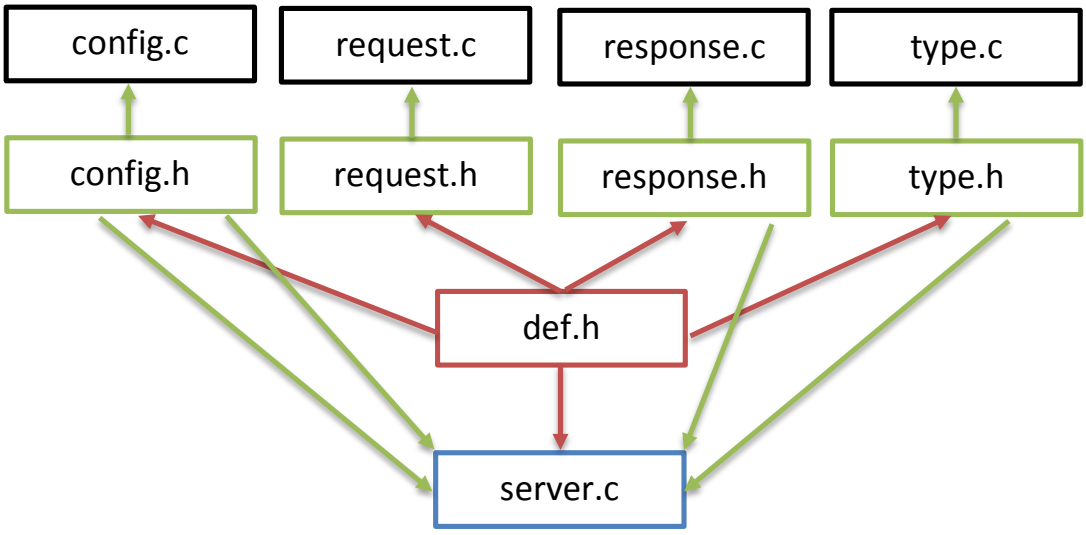
Θα πρέπει να χειρίζεται αιτήσεις των μεθόδων **GET**, **HEAD** και **DELETE**. Αν σε μια αίτηση ζητηθεί άλλη μέθοδος που δεν είναι υλοποιημένη πρέπει να απαντάμε με το ανάλογο μήνυμα (501 Not Implemented). Στο μήνυμα απάντησης θα πρέπει να συμπεριλαμβάνονται οι ακόλουθες επικεφαλίδες: **Connection**, **Server**, **Content-Length** και **Content-Type**.

Ο server θα πρέπει να είναι πολυνηματικός έτσι ώστε να είναι σε θέση να εξυπηρετεί ταυτόχρονα πολλές αιτήσεις από πελάτες. Επίσης οι παράμετροι ρυθμίσεις του server θα πρέπει να βρίσκονται σε ένα αρχείο config.txt.

Περιγραφή Λύσης/Αρχιτεκτονική Συστήματος:

Η αρχή του προγράμματος (main) γίνεται από το server.c. Χρησιμοποιώντας το config.c, ο server αρχικοποιήται βάση του αρχείο config.txt και εκτυπώνεται το configuration στην οθόνη. Στην συνέχεια, ανοίγει socket στην καθορισμένη από το config.txt πόρτα και περιμένει πελάτες. Με το που συνδέεται κάποιος μαζί του τότε του αναθέτει νήμα και handler από το request.c.

Στην συνέχεια ανάλογα με την μέθοδο που επιθυμεί ο πελάτες γίνεται και το ανάλογο response από τις συναρτήσεις στο response.c. Η σύνδεση των .c/.h αρχείων από το makefile γίνεται ως εξής:



Κώδικας:

server.c:

```
/**
 * @file server.
 * @brief Multithreaded HTTP1.1 Server in C
 *
 * This is the solution of the exercise 4
 * (EPL371-2015), involving creating a
 * multithreaded HTTP server in C language.
 *
 * @author Andreas Andreou(ID: 983759)
 */

#include "def.h"
#include "config.h"
#include "types.h"
#include "request.h"
#include "response.h"

/**
 * @brief Program's entry point.
 *
 * This is the entry point of the server.
 *
 * @param argc Length of arguments.
 * @param argv Arguments table.
 * @return Nothing important.
 */
int main(int argc, char *argv[]) {

	// Variables
	int sock, newsock, serverlen, *new_sock, clientlen;
	struct sockaddr_in server, client;
	struct sockaddr *serverptr, *clientptr;
	struct hostent *rem;

	// Set and print server's configuration
	setConfig();
	printConfig();

	// Create socket
	if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
		perror("socket");
		exit(1);
	}

	//Prepare the sockaddr_in structure
	server.sin_family = AF_INET; // Internet domain
	server.sin_addr.s_addr = htonl(INADDR_ANY); // My Internet address
	server.sin_port = htons(conf.port); // The config port
	serverptr = (struct sockaddr *) &server;
	serverlen = sizeof(server);

	// Bind socket to an address
	if (bind(sock, serverptr, serverlen) < 0) {
		perror("bind");
		exit(1);
	}

	// Listen for connections
```

```
	if (listen(sock, 3) < 0) {
		perror("listen");
		exit(1);
	}

	//Accept and incoming connection
	printf("Listening for connections to port %d\n", conf.port);
	while (1) {

		// Accept connection
		clientptr = (struct sockaddr *) &client;
		clientlen = sizeof(client);
		if ((newsock = accept(sock, clientptr, (socklen_t*) &clientlen)) < 0) {
			perror("accept failed");
			return 1;
		}

		// Find client's address
		rem = gethostbyaddr((char *) &client.sin_addr.s_addr,
				sizeof(client.sin_addr.s_addr), client.sin_family);
		if (rem == NULL) {
			perror("gethostbyaddr");
			exit(1);
		}
		printf("Accepted connection from %s\n", rem->h_name);

		/* Create child for serving the client */
		switch (fork()) {
		case -1: {
			perror("fork");
			exit(1);
		}
		case 0: {
			pthread_t sniffer_thread;
			new_sock = malloc(1);
			*new_sock = newsock;

			if (pthread_create(&sniffer_thread, NULL, connection_handler,
					(void*) new_sock) < 0) {
				perror("could not create thread");
				exit(1);
			}
			printf("Handler assigned\n");
		}
		}
	}
	return 0;
}
```

def.h:

```c
#ifndef DEF_H_
#define DEF_H_

// Libraries
#include <sys/types.h>   // For sockets
#include <sys/socket.h>  // For sockets
#include <netinet/in.h>  // For Internet sockets
#include <netdb.h>       // For get host by address
#include <pthread.h>     // For threading
#include <stdio.h>       // For I/O
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

// Global Constants
#define FCONFIG "config.txt"
#define MAXBUF 1024
#define DELIM "="

// Structs
struct config {
	int num_treads;
	int port;
	char name[MAXBUF];
};
struct streq {
	char method[15]; // GET | HEAD | DELETE
	char path[256];   // path to send
	char con[15];    // keep-alive | close
};

// Global Variables
struct config conf;

#endif /* DEF_H_ */
```

types.h:

```c
#include "def.h"

#ifndef TYPES_H_
#define TYPES_H_

/**
 * Returns the extension of the filename or null.
 * @param filename Filename to find it's extension.
 * @return Filename's extension or null.
 */
char *get_filename_ext(char *filename);

/**
 * By using the get_filename_ext, this function
 * returns the content_type of the given filename.
 * @param filename Filename to find it's content-type.
 * @return Filename's content-type.
 */
char *get_content_type(char *filename);

#endif /* TYPES_H_ */
```

types.c:

```
#include "types.h"

char *get_filename_ext(char *filename) {
	char *dot = strrchr(filename, '.');
	if (!dot || dot == filename)
		return "";
	return dot + 1;
}

char *get_content_type(char *filename) {
	char *ext = get_filename_ext(filename);

	if (strcmp(ext, "txt") == 0 || strcmp(ext, "sed") == 0
			|| strcmp(ext, "awk") == 0 || strcmp(ext, "c") == 0
			|| strcmp(ext, "h") == 0) {
		return ("text/plain");
	}

	if (strcmp(ext, "html") == 0 || strcmp(ext, "htm") == 0) {
		return ("text/html");
	}

	if (strcmp(ext, "jpeg") == 0 || strcmp(ext, "jpg") == 0) {
		return ("image/jpeg");
	}

	if (strcmp(ext, "png") == 0) {
		return ("image/png");
	}

	if (strcmp(ext, "css") == 0) {
		return ("text/css");
	}

	if (strcmp(ext, "js") == 0) {
		return ("text/javascript");
	}

	if (strcmp(ext, "gif") == 0) {
		return ("image/gif");
	}

	if (strcmp(ext, "pdf") == 0) {
		return ("application/pdf");
	}

	return "application/octet-stream";
}
```

config.h:

```
#include "def.h"

#ifndef CONFIG_H_
#define CONFIG_H_

/**
 * Sets the configurations of the server base on the
 * filename in 'FCONFIG' constant.
 */
void setConfig();

/**
 * Prints the configuration information of the server.
 * The configuration includes: Number of threads, port
 * number and the server's name.
 */
void printConfig();

#endif /* CONFIG_H_ */
```

config.c:

```
#include "config.h"

void setConfig() {
	FILE *file = fopen(FCONFIG, "r");

	if (file == NULL) {
		printf("*** Unable to configure server! System will now exit. ***\n");
		exit(0);
	}
	if (file != NULL) {
		char line[MAXBUF];
		int i = 0;

		while (fgets(line, sizeof(line), file) != NULL) {
			char *cfline;
			cfline = strstr((char *) line, DELIM);
			cfline = cfline + strlen(DELIM);

			if (i == 0) {
				char temp_treads[MAXBUF];
				memcpy(temp_treads, cfline, strlen(cfline));
				conf.num_treads = atoi(temp_treads);
			} else if (i == 1) {
				char temp_port[MAXBUF];
				memcpy(temp_port, cfline, strlen(cfline));
				conf.port = atoi(temp_port);
			} else if (i == 2) {
				memcpy(conf.name, cfline, strlen(cfline));
			}
			i++;
		}
	}
	fclose(file);
}

void printConfig() {
	printf("SERVER CONFIGURATION\n");
```

```
	printf("====================\n");
	printf("Number of Threads: %d\n", conf.num_treads);
	printf("Port Number: %d\n", conf.port);
	printf("Server Name: %s\n", conf.name);
	printf("------------------------\n\n");
}
```

request.h:

```
#include "def.h"

#ifndef REQUEST_H_
#define REQUEST_H_

/**
 * Handles connection for each client
 * @param socket_desc
 */
void *connection_handler(void *socket_desc);

/**
 * Reads the client's request from the given
 * socket and fills the given request struct.
 * @param sock Socket to read request from.
 * @param r Request struct to break up the request.
 */
void getRequest(int sock, struct streq *r);

#endif /* REQUEST_H_ */
```

request.c:

```
#include "request.h"

void *connection_handler(void *socket_desc) {

	int sock = *(int*) socket_desc; //Get the socket descriptor
	struct streq req;

	// While connection: keep-alive
	do {
		// Read the Request
		getRequest(sock, &req);

		// Response by method
		if (strcmp(req.method, "GET") == 0)
			responseGet(sock, req);
		else if (strcmp(req.method, "HEAD") == 0)
			responseHead(sock, req);
		else if (strcmp(req.method, "DELETE") == 0)
			responseDelete(sock, req);
		else
			sentResponse(sock, req.con, "501 Not Implemented", "text/plain",
					"Method not implemented!\r\n");

	} while (strcmp(req.con, "keep-alive") == 0);
	printf("Client disconnected\n");

	//Free the socket pointer
	free(socket_desc);
```

```
	return 0;
}

void getRequest(int sock, struct streq *r) {
	char buf[512];
	bzero(buf, sizeof(buf)); // Initialize buffer

	// Read from socket
	if (read(sock, buf, sizeof(buf)) < 0) {
		perror("read");
		exit(1);
	}

	// Tokenize Headers
	char *line = strtok(buf, " \r\n");

	// Get Request method
	strcpy(r->method, line);
	line = strtok(NULL, " \r\n");

	// Get Request path
	strcpy(r->path, "anyplace");
	strcat(r->path, line);
	if (strcmp(line, "/") == 0) {
		strcat(r->path, "index.html");
	}
	line = strtok(NULL, " \r\n");

	// Get Request Connection
	while (line != NULL) {
		if (strcmp(line, "Connection:") == 0) {
			line = strtok(NULL, " \r\n");
			strcpy(r->con, line);
		}
		line = strtok(NULL, " \r\n");
	}
	printf("------------------------------------------------\n");
	printf("Request: %s %s (%s)\n", r->method, r->path, r->con);
}
```

response.h:

```
#include "def.h"

#ifndef RESPONSE_H_
#define RESPONSE_H_

/**
 * The functions builds the appropriate response base on the given
 * arguments and writes on the given socket.
 * @param sock Given socket to response to.
 * @param con The connection that the client wants(keep-alive/close)
 * @param Status_code Status Code for the response (e.g. "200 OK").
 * @param Content_Type Content type for the response (e.g. "text/plain").
 * @param HTML The raw content for the response.
 */
void sentResponse(int sock, char *con, char *Status_code, char *Content_Type,
             char *HTML);


/**
 * The function response to the GET request of
 * a client to the given socket.
 * @param sock Socket to response to the client.
 * @param r Request struct from the client.
 */
void responseGet(int sock, struct streq r);

/**
 * Checks if the requested path exists. If it does and it's a
 * file, it sends 200 OK, if the path is a directory, it sends
 * @param sock Given socket to response to.
 * @param r
 */
void responseHead(int sock, struct streq r);


/**
 * Tries to delete a file. It response as follows:
 * Exists & Is File & Deleted: 200.
 * Exists & Is File & Not Deleted: 500.
 * Exists & Is Folder: 403.
 * Not Exists: 404.
 * @param sock Given socket to response to.
 * @param r Request struct from the client.
 */
void responseDelete(int sock, struct streq r);

#endif /* RESPONSE_H_ */
```

response.c:

```
#include "response.h"

void sentResponse(int sock, char *con, char *Status_code, char *Content_Type,
		char *HTML) {
	// Set Response Headers
	char *head = "HTTP/1.1 ";
	char *server_head = "\r\nServer: ";
	char *length_head = "\r\nContent-Length: ";
	char *connection_head = "\r\nConnection: ";
	char *content_head = "\r\nContent-Type: ";
	char *newline = "\r\n";
	char Content_Length[100];
	int content_length = strlen(HTML);
	char connection[100];
	if (strcmp(con, "keep-alive") == 0)
		sprintf(connection, "%s", "keep-alive");
	else
		sprintf(connection, "%s", "close");

	// Malloc for response message
	sprintf(Content_Length, "%i", content_length);
	char *message = malloc(
			(strlen(head) + strlen(Status_code) + strlen(server_head)
					+ strlen(conf.name) + strlen(length_head)
					+ strlen(Content_Length) + strlen(connection_head)
					+ strlen(connection) + strlen(content_head)
					+ strlen(Content_Type) + strlen(newline) + content_length
					+ sizeof(char)) * 2);

	// Built response message
	if (message != NULL) {
		strcpy(message, head);
		strcat(message, Status_code);
		strcat(message, server_head);
		strcat(message, conf.name);
		strcat(message, length_head);
		strcat(message, Content_Length);
		strcat(message, connection_head);
		strcat(message, connection);
		strcat(message, content_head);
		strcat(message, Content_Type);
		strcat(message, newline);
		strcat(message, newline);
		strcat(message, HTML);

		// Send message
		if (write(sock, message, strlen(message)) < 0) {
			perror("write");
			exit(1);
		}
		free(message);
	}
	printf("Response: %s %s (%s)\n", head, Status_code, connection);
}

void responseGet(int sock, struct streq r) {
	struct stat s;
	int exists = stat(r.path, &s);
	if (exists == -1) {
		sentResponse(sock, r.con, "404 Not Found", "text/html",
```

```
                        "<h1>HTTP Error 404</h1>");
        } else {
                // Directory listing denied, show 403
                if (S_ISDIR(s.st_mode)) {
                        sentResponse(sock, r.con, "403 Forbidden", "text/plain", "");
                } else {
                        // Get file's content-type
                        char *ctype = get_content_type(r.path);

                        // Get file's content
                        char *buffer = 0;
                        long length;
                        FILE * f = fopen(r.path, "rb");

                        if (f) {
                                fseek(f, 0, SEEK_END);
                                length = ftell(f);
                                fseek(f, 0, SEEK_SET);
                                buffer = malloc(length);
                                if (buffer) {
                                        fread(buffer, 1, length, f);
                                }
                                fclose(f);
                        }
                        sentResponse(sock, r.con, "200 OK", ctype, buffer);
                }
        }
}

void responseHead(int sock, struct streq r) {
        struct stat s;
        int exists = stat(r.path, &s);
        if (exists == -1) {
                sentResponse(sock, r.con, "404 Not Found", "text/plain", "");
        } else {
                // Directory listing denied, show 403
                if (S_ISDIR(s.st_mode)) {
                        sentResponse(sock, r.con, "403 Forbidden", "text/plain", "");
                } else {
                        sentResponse(sock, r.con, "200 OK", "text/plain", "");
                }
        }
}
```

```
void responseDelete(int sock, struct streq r) {
	struct stat s;
	int exists = stat(r.path, &s);
	if (exists == -1) {
		sentResponse(sock, r.con, "404 Not Found", "text/plain", "");
	} else {
		// Directory delete denied, show 403
		if (S_ISDIR(s.st_mode)) {
			sentResponse(sock, r.con, "403 Forbidden", "text/plain", "");
		} else {
			// try to delete requested file
			int status = remove(r.path);
			if (status == 0) {
				sentResponse(sock, r.con, "200 OK", "text/plain",
						"File Deleted\n");
			} else {
				sentResponse(sock, r.con, "500 Internal Error", "text/plain",
						"Unable to delete the file\n");
			}
		}
	}
}
```

Εκτέλεση:

Eclipse:

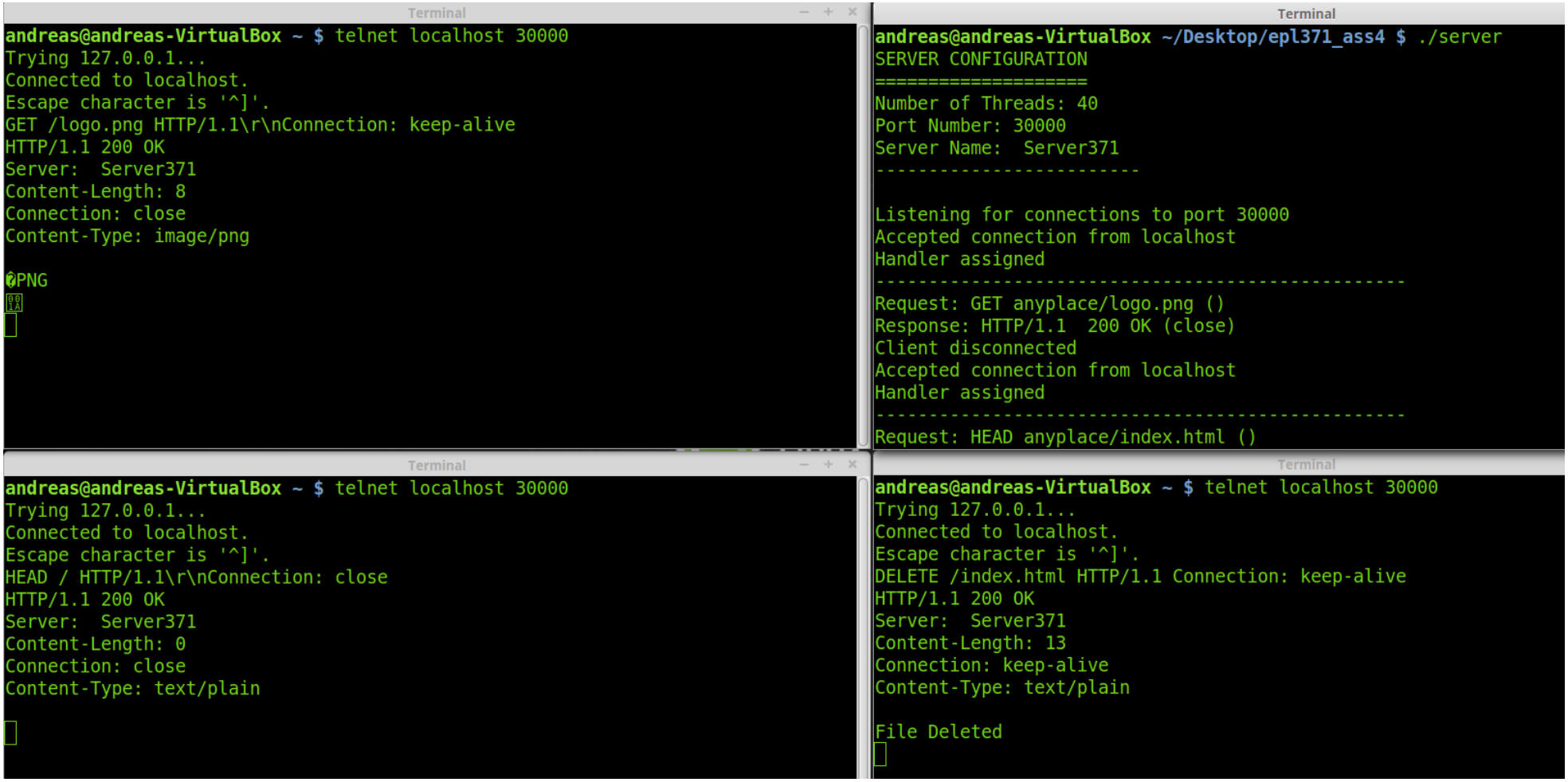
To add pthread library to project flow these steps:

right click on the project in the project explorer
properties -> c/c++ build -> Settings -> linker -> libraries -> add -> pthread -> ok -> rebuild

Terminal:

```
Make all
./server
```

Output:


Terminal
andreas@andreas-VirtualBox ~ $ telnet localhost 30000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /logo.png HTTP/1.1\r\nConnection: keep-alive
HTTP/1.1 200 OK
Server: Server371
Content-Length: 8
Connection: close
Content-Type: image/png
PNG
Terminal
andreas@andreas-VirtualBox ~/Desktop/epl371_ass4 $ ./server
SERVER CONFIGURATION
Number of Threads: 40
Port Number: 30000
Server Name: Server371
Listening for connections to port 30000
Accepted connection from localhost
Handler assigned
Request: GET anyplace/logo.png ()
Response: HTTP/1.1 200 OK (close)
Client disconnected
Accepted connection from localhost
Handler assigned
Request: HEAD anyplace/index.html ()
Terminal
andreas@andreas-VirtualBox ~ $ telnet localhost 30000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
HEAD / HTTP/1.1\r\nConnection: close
HTTP/1.1 200 OK
Server: Server371
Content-Length: 0
Connection: close
Content-Type: text/plain
Terminal
andreas@andreas-VirtualBox ~ $ telnet localhost 30000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
DELETE /index.html HTTP/1.1 Connection: keep-alive
HTTP/1.1 200 OK
Server: Server371
Content-Length: 13
Connection: keep-alive
Content-Type: text/plain
File Deleted