**Team5Lab3Contents.docx**

**server.c**

#include <arpa/inet.h>

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h> // string manipulation

#include <sys/socket.h>

#include <unistd.h> // read, write

#define BACKLOG 10 // queue length

#define MAX\_HEADER\_SIZE 1024

#define MAX\_DATA\_SIZE 2048

struct client\_param {

int client\_sd;

char client\_ip[32];

};

int start\_socket(int port);

void \*conn\_handler(void \*param);

int get\_str\_until\_space(char \*src, int src\_start, char \*dst);

int load\_file\_to\_buffer(char \*file\_name, char \*buffer);

int main() {

int sd = start\_socket(5678);

// Define client variables

struct sockaddr\_in client;

socklen\_t client\_len = sizeof(struct sockaddr\_in);

while (1) {

// Accept

int client\_sd = accept(sd, (struct sockaddr \*)&client, &client\_len);

char client\_ip[32] = {0};

if (client\_sd < 0) {

printf("Accept failed\n");

exit(-1);

}

// Format client ip address and port into client\_ip

sprintf(client\_ip, "%s:%d", inet\_ntoa(client.sin\_addr), client.sin\_port);

printf("Connection accepted with client %s\n", client\_ip);

pthread\_t conn\_thread;

struct client\_param param;

param.client\_sd = client\_sd;

strcpy(param.client\_ip, client\_ip);

if (pthread\_create(&conn\_thread, NULL, conn\_handler, &param) < 0) {

printf("Create thread failed");

exit(-1);

}

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: start\_socket

\* Create a socket, bind and listen

\* Parameters:

\* port: listening port

\* Returns:

\* sd (socket descriptor)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int start\_socket(int port) {

// Create socket

int sd = socket(PF\_INET, SOCK\_STREAM, 0); // PF\_INET for BSD

if (sd < 1) {

printf("Couldn't create socket\n");

exit(-1);

}

// Set server configuration

struct sockaddr\_in server;

server.sin\_family = AF\_INET;

server.sin\_addr.s\_addr = INADDR\_ANY; // enable any ip\_addr

server.sin\_port = htons(port);

// Bind

if (bind(sd, (struct sockaddr \*)&server, sizeof(server)) < 0) {

printf("Bind failed\n");

exit(-1);

}

// Listen

if (listen(sd, BACKLOG) < 0) {

printf("Listen failed\n");

exit(-1);

}

printf("HTTP Server running on port %d\n", port);

return sd;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: conn\_handler

\* Thread executing function, read and write with client by given client\_sd

\* Parameters:

\* void \*param - a client\_param structure

\* Returns:

\* 0 - successful

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void \*conn\_handler(void \*param) {

struct client\_param \*cp = (struct client\_param \*)param;

int client\_sd = cp->client\_sd;

char\* client\_ip = cp->client\_ip;

// Read

char client\_header[MAX\_HEADER\_SIZE] = {0};

int read\_size = read(client\_sd, client\_header, MAX\_HEADER\_SIZE);

if (read\_size < 0) {

printf("Read failed\n");

return (void \*) -1;

}

printf("Client header ================\n");

printf("%s", client\_header);

printf("==============================\n");

// find out HTTP method and request file

char method[8], file\_name[32];

int pos = get\_str\_until\_space(client\_header, 0, method); // find method

pos = get\_str\_until\_space(client\_header, pos + 2, file\_name); // find file\_name

printf("Method = %s, File\_name = %s\n", method, file\_name);

// Respond client according given method and file name

char res[MAX\_HEADER\_SIZE] = {0}, buffer[MAX\_DATA\_SIZE] = {0};

if (strcmp(method, "GET") == 0) { // send file

int file\_size = load\_file\_to\_buffer(file\_name, buffer);

if (file\_size == -1) { // couldn't find file or other errors

// 404 Not Found

strcpy(res, "HTTP/1.1 404 Not Found\r\n\r\n");

if (write(client\_sd, res, strlen(res)) < 0) {

printf("Write failed\n");

return (void \*) -1;

}

// Transfer Error Page

if (strcmp(file\_name, "") != 0) {

sprintf(res, "<!doctype html><html><body>\

<h1>404 Not Found, File %s might not exist</h1>\

</body></html>\r\n\r\n", file\_name);

} else {

sprintf(res, "<!doctype html><html><body>\

<h1>404 Not Found, Please request by given file name</h1>\

</body></html>\r\n\r\n");

}

if (write(client\_sd, res, strlen(res)) < 0) {

printf("Write failed\n");

return (void \*) -1;

}

} else {

// 200 OK

strcpy(res, "HTTP/1.1 200 OK\r\n\r\n");

if (write(client\_sd, res, strlen(res)) < 0) {

printf("Write failed\n");

return (void \*) -1;

}

// Transfer data

strcpy(res, "The Second Write\r\n\r\n");

if (write(client\_sd, buffer, file\_size) < 0) {

printf("Write failed\n");

return (void \*) -1;

}

}

} else if (strcmp(method, "POST") == 0 || strcmp(method, "PUT") == 0) {

// 403 Forbidden

strcpy(res, "HTTP/1.1 403 Forbidden\r\n\r\n");

if (write(client\_sd, res, strlen(res)) < 0) {

printf("Write failed\n");

return (void \*) -1;

}

// Transfer Error Page

sprintf(res, "<!doctype html><html><body>\

<h1>403 Forbidden</h1>\

</body></html>\r\n\r\n");

if (write(client\_sd, res, strlen(res)) < 0) {

printf("Write failed\n");

return (void \*) -1;

}

} else {

// 400 Bad Request

strcpy(res, "HTTP/1.1 400 Bad Request\r\n\r\n");

if (write(client\_sd, res, strlen(res)) < 0) {

printf("Write failed\n");

return (void \*) -1;

}

// Transfer Error Page

sprintf(res, "<!doctype html><html><body>\

<h1>400 Bad Request</h1>\

</body></html>\r\n\r\n");

if (write(client\_sd, res, strlen(res)) < 0) {

printf("Write failed\n");

return (void \*) -1;

}

}

printf("Write Response to %s\n\n", client\_ip);

close(client\_sd);

return (void \*) 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: get\_str\_until\_space

\* Get substr of src from given position until the first space and put in dst

\* EX - get\_str\_until\_space("abc def", 0, dst) return 3, dst will be "abc"

\* Parameters:

\* src - original string

\* src\_start - the start position (including) of src

\* dst - the buffer that the substring will be put into

\* Returns:

\* the position of next space or the length of src (it traverses to the end)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int get\_str\_until\_space(char \*src, int src\_start, char \*dst) {

int i;

for (i = 0; src\_start < strlen(src) && src[src\_start] != ' '; i++) {

dst[i] = src[src\_start];

src\_start++;

}

dst[i] = '\0';

return src\_start;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: load\_file\_to\_buffer

\* Read file by a given name

\* Parameters:

\* file\_name - file name, in the same path that server is executed

\* buffer - the buffer that file content will put into

\* Returns:

\* >= 0 - file size

\* -1 - error

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int load\_file\_to\_buffer(char \*file\_name, char \*buffer) {

FILE \*file;

file = fopen(file\_name, "rb");

if (!file) {

printf("Couldn't find %s\n", file\_name);

return -1;

}

fread(buffer, MAX\_DATA\_SIZE, 1, file);

fseek(file, 0L, SEEK\_END); // in order to find file size

int file\_size = ftell(file);

printf("Server read %s, size %d bytes\n", file\_name, file\_size);

fclose(file);

return file\_size;

}

**client.c**

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h> // string manipulation

#include <sys/socket.h>

#include <unistd.h> // read, write

#define PORT 5678

#define MSG\_SIZE 2048

int main() {

// User input file\_name

char file\_name[128];

printf("Please input file name to request: ");

scanf("%s", file\_name);

// Create socket

int sd = socket(PF\_INET, SOCK\_STREAM, 0); // PF\_INET for BSD

if (sd < 1) {

printf("Couldn't create socket\n");

return -1;

}

printf("TCP socket created\n");

struct sockaddr\_in server;

server.sin\_family = AF\_INET;

server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server.sin\_port = htons(PORT);

// Connect

if (connect(sd, (struct sockaddr \*)&server, sizeof(server)) < 0) {

printf("Connect failed\n");

return -1;

}

printf("Connect successful\n");

// Write

// Define message to be written

char client\_msg[MSG\_SIZE] = {0};

sprintf(client\_msg, "GET /%s HTTP/1.1\r\n\r\n", file\_name);

if (write(sd, client\_msg, strlen(client\_msg)) < 0) {

printf("Write failed\n");

return -1;

}

printf("Write successful\n");

// Read

// Define message buffer from server

char server\_msg[MSG\_SIZE] = {0};

int read\_size = read(sd, server\_msg, MSG\_SIZE);

if (read\_size < 0) {

printf("Read failed\n");

return -1;

}

printf("Read from server:\n%s\n", server\_msg);

char file\_from\_server[MSG\_SIZE] = {0};

read\_size = read(sd, file\_from\_server, MSG\_SIZE);

if (read\_size < 0) {

printf("Read failed\n");

return -1;

}

if (strcmp(server\_msg, "HTTP/1.1 200 OK\r\n\r\n") == 0) {

// write file

FILE \*file\_to\_write;

file\_to\_write = fopen(file\_name, "wb");

fwrite(file\_from\_server, read\_size, 1, file\_to\_write);

fclose(file\_to\_write);

} else {

printf("Fail message from server:\n%s\n", file\_from\_server);

}

return 0;

}