ECE Paris

Computer Networks

ING 4 – SI (Information Systems, Big Data and Cyber Security)

Rafik ZITOUNI: rafik.zitouni@ece.fr

Takvor MAGADIS: tmagadis@inseec-edu.com

This lab was inspired from the works of X. LI and F. Fauberteau

LAB 3

Contents

I. Objective

1. Socket network programming
2. Review client/server approach.

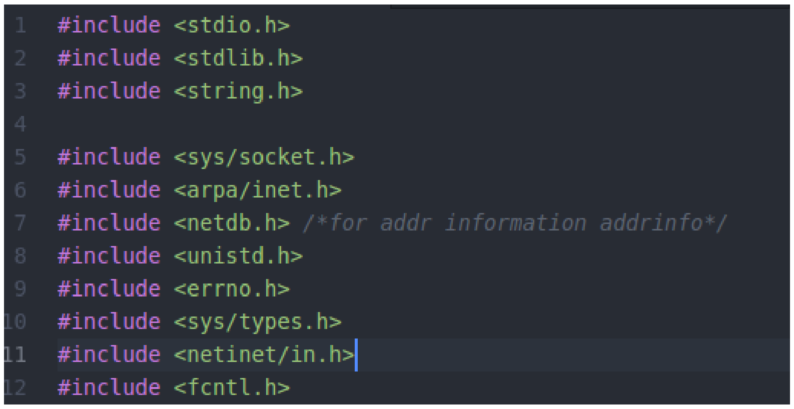
- Each group (composed of **3 students** at most) shall submit a report **in campus.**
- The report (**PDF format is accepted**) shall be uploaded on the campus page before the deadline **November 22 at 23h55.**
- The source code and the exec files of the server (websrv.c and websrv) as well as the index.html should be zipped in one file. Please propose a small html file.

II. Web server

In this lab, you will write a very simple Web server in C language. The connection between your server and a web browser will be made by a network socket over the TCP/IP protocol. Throughout different section you will build the server program using the different chunks of the source code.

1. Skeleton

You can start with any text editor (prefer the one you are able to use) by coding a main function and import libraries . For example, the file name could be **websrv.c**.

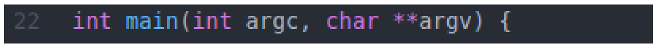


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h> /*for addr information addrinfo*/
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <fcntl.h>
```

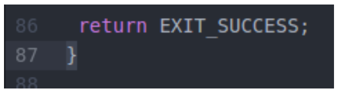
Q1: explain why we are using the library <netinet/in.h >

The source head code of the main would be present as follow:



```
int main(int argc, char **argv) {
```

Q2: Add the following line, compile and execute the server code. What is the required Linux command line and the name of the Compiler.



```
  return EXIT_SUCCESS;
}
```

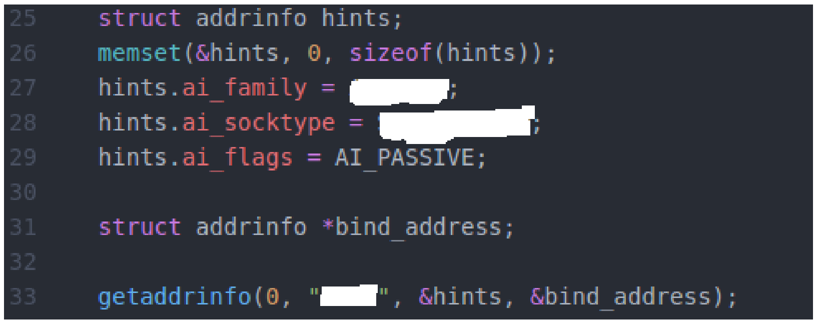
2. Configuring local address and creating the socket

These are the two first steps in your server program. We start by configuring the socket setting giving local address and then creating a server socket. The command **man 2 socket** gives you more information about this function.

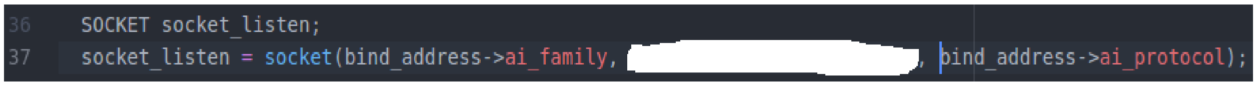
int socket(**int** domain, **int** type, **int** protocol);

The domain corresponds to the network layer protocol (layer 3) and can take the values AF_UNIX for local communications, **AF_INET** for IPv4 protocols and **AF_INET6** for IPv6 protocols, among others.

Q3: What would be the suitable domain in our case the domain, the type and the port number?



```
25    struct addrinfo hints;
26    memset(&hints, 0, sizeof(hints));
27    hints.ai_family =          ;
28    hints.ai_socktype =            ;
29    hints.ai_flags = AI_PASSIVE;
30
31    struct addrinfo *bind_address;
32
33    getaddrinfo(0, "     ", &hints, &bind_address);
```



```
36   SOCKET socket_listen;
37   socket_listen = socket(bind_address->ai_family,                     , bind_address->ai_protocol);
```

Q4: Complete the instruction of line 37.

The variable **socket_listen** represents the socket file descriptor you will use for this entire exercise. All the functions of this exercise return -1 when they fail. You must test this return value to avoid to write a quiet program that silently crashes. In order to make that test, we define for you a macro. This macro acts as a function and you can give it the name of the faulty socket function as a parameter.

3. Socket binding

The third step is to bind the socket to an IP address in order to make it reachable from the IP network.

```
bind(socket_listen, bind_address->ai_addr, bind_address->ai_addrlen);
```

The parameter **socket_listen** is always your very same socket file descriptor.

The parameter addr is a pointer on the address you previously declared. You can notice the type of the parameter is not exactly the same as the one used. Actually, it allows to make generic code (like with inheritance in Java).

Q5: Add the source code to manage the error of the binding function.

4. Socket listening

The fourth step is to make the socket listen to a new input connection. The command man 2 listen gives you the following information:

```
int listen(int socket_listen, int backlog);
```

The parameter backlog is useless in your case and you can safely set it to 0.

Q6: Here you should add your code to manage the error of socket listening.

5. Socket acceptance

The fifth step is to make the socket accept new connections. The command man 2 accept gives you the following information:

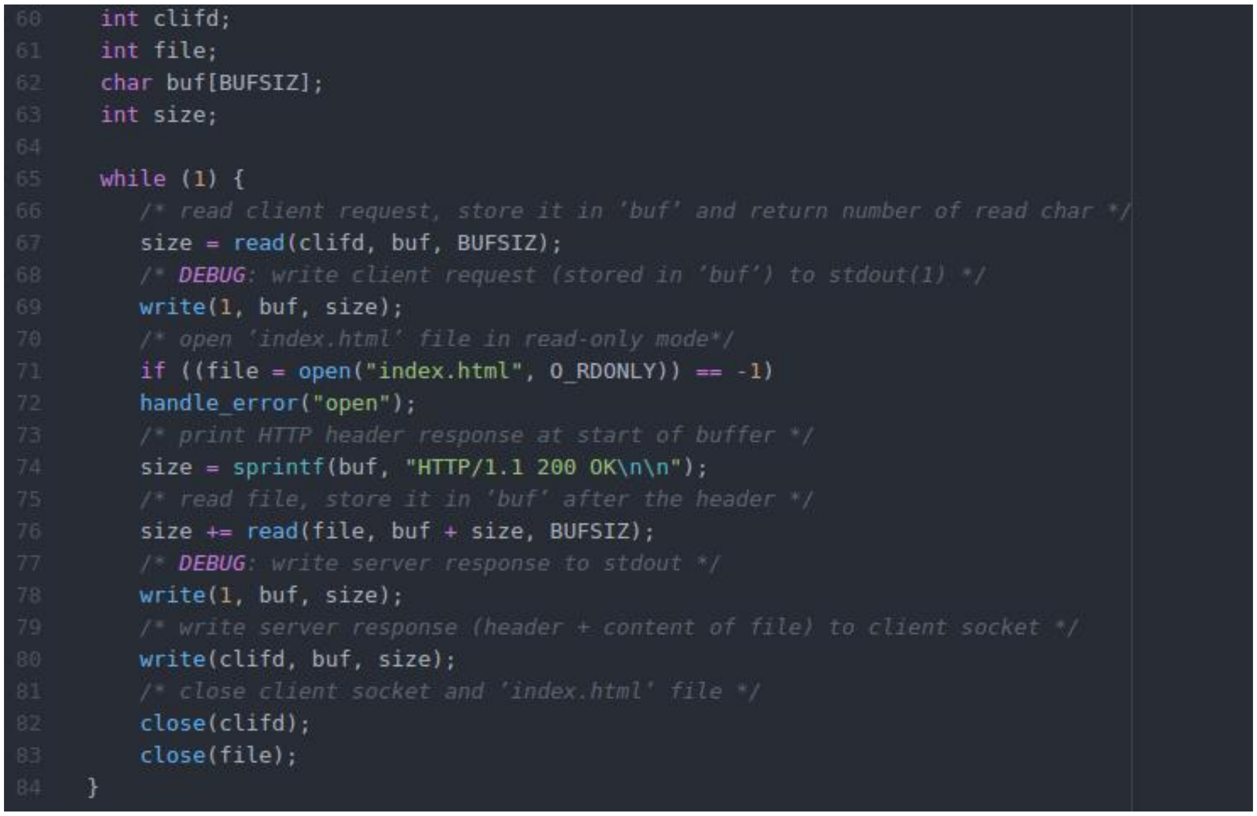
```
SOCKET socket_client = accept(socket_listen, NULL, NULL);
```

The fields addr and addrlen are for the address of the client socket (from the distant machine) accepted for this connection. At this step, you don't need of this address and you can safely pass NULL to these two parameters. The function accept is a system call that blocks your program until a new connection intends to establish.

III. Connection test

When you try to connect to your server program through the network (from another machine) using a web browser, your program quits. Actually, the connection has been accepted, and the function accept returns the client socket file descriptor. Your server can use it to send a message to the client. Therefore, you must store it in a new variable (*e.g.* **clifd**).

Q7: Now, you will surround your accept function call by an infinite while loop (a server program should never quit until it is interrupted). Add the following code to make your server program able to serve an "index.html" file through the HTTP protocol. The "index.html" could be the simple html file.



```c
  int clifd;
  int file;
  char buf[BUFSIZ];
  int size;

  while (1) {
     /* read client request, store it in 'buf' and return number of read char */
     size = read(clifd, buf, BUFSIZ);
     /* DEBUG: write client request (stored in 'buf') to stdout(1) */
     write(1, buf, size);
     /* open 'index.html' file in read-only mode*/
     if ((file = open("index.html", O_RDONLY)) == -1)
     handle_error("open");
     /* print HTTP header response at start of buffer */
     size = sprintf(buf, "HTTP/1.1 200 OK\n\n");
     /* read file, store it in 'buf' after the header */
     size += read(file, buf + size, BUFSIZ);
     /* DEBUG: write server response to stdout */
     write(1, buf, size);
     /* write server response (header + content of file) to client socket */
     write(clifd, buf, size);
     /* close client socket and 'index.html' file */
     close(clifd);
     close(file);
  }
```

Q8 (Bonus) : The last objective is to implement the send function to share html page with the client.