Barbe Victor
Gaucher Pierre-louis

Laboratory 3 – Sockets and network programming

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 (only PDF format is accepted) shall be uploaded on the campus page.

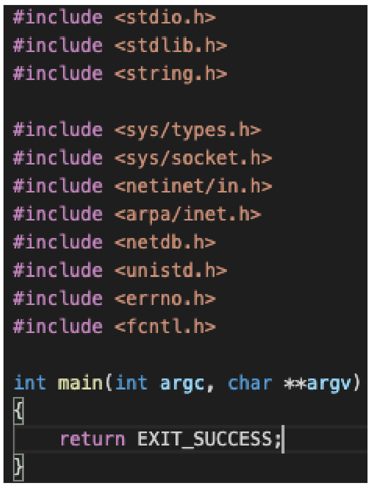
II. Web server

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

For this code we are using the library <netinet/in.h > because this header contains definitions for the internet protocols we will have to use in this lab, and we need it for the socket usage.

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

Here is the code we have, including all the libraries and the simple main.

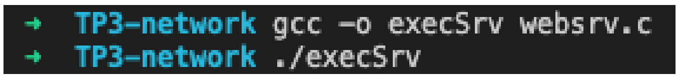


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>

int main(int argc, char **argv)
{
    return EXIT_SUCCESS;
}
```

To execute this code, we need to use the following commands in a terminal:



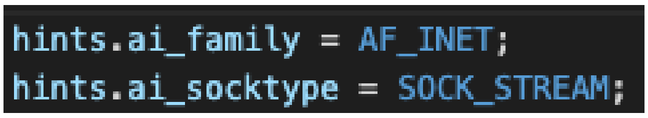
```
➜  TP3-network gcc -o execSrv websrv.c
➜  TP3-network ./execSrv
```

The first command will create an executable of the program websrv.c which will be called execSrv using GCC. The second command will execute the executable file.

The name of the compiler is GCC.

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

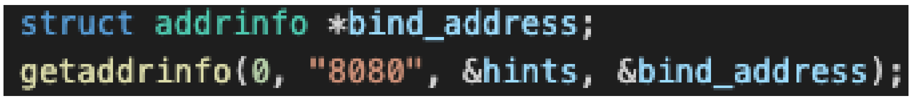
Here is the part of the source code where we define the domain, type, and port number.



```
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
```

We use AF_INET as the domain because we want to use IPV4 addressing. Then we define the type as SOCK_STREAM because we will be using TCP for this program because it is a connection-oriented protocol.

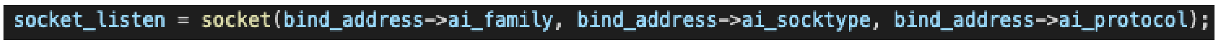
As for the port number we will use 8080 as defined during the creation of the structure:



```
struct addrinfo *bind_address;
getaddrinfo(0, "8080", &hints, &bind_address);
```

Q4: Complete the instruction of line 37.

First, we define the socket as shown on line 36: we define it as a SOCKET type and the Macro defines it as int. Then, we completed the line 37 like this:

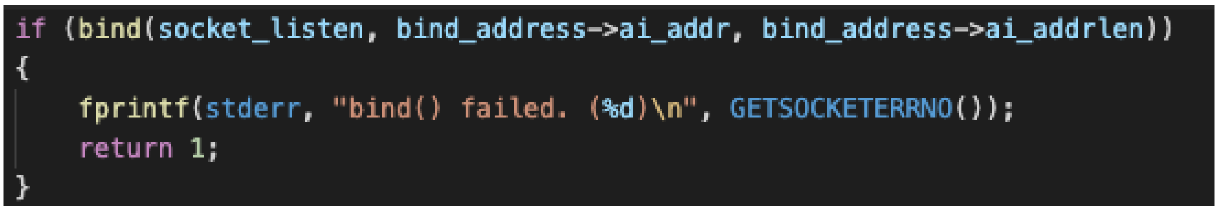


```
socket_listen = socket(bind_address->ai_family, bind_address->ai_socktype, bind_address->ai_protocol);
```

As seen above the socket takes as a parameter a domain, a type, and a protocol. These are defined for this socket using bind_address, which is a pointer to the addrinfo structure we created earlier. bind_address contains the info of getaddrinfo().

Using this source Code, we allocated to this socket the suited parameters for the domain, type and port number we discussed in question 3.

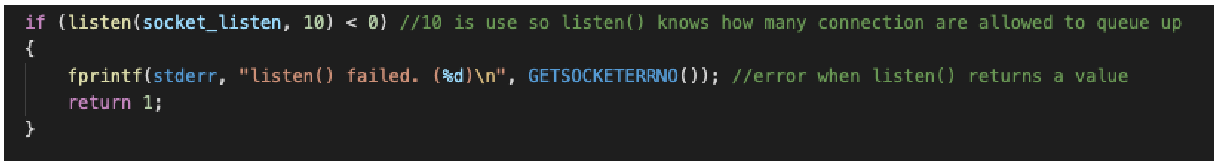
Q5: Add the source code to manage the error of the binding function.



```
if (bind(socket_listen, bind_address->ai_addr, bind_address->ai_addrlen))
{
    fprintf(stderr, "bind() failed. (%d)\n", GETSOCKETERRNO());
    return 1;
}
```

To manage the error of the binding function, we use this source code. Using the Macro GETSOCKET ERRNO we defined at the beginning, we return an error message and exit the program if an error occurs on the binding function.

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



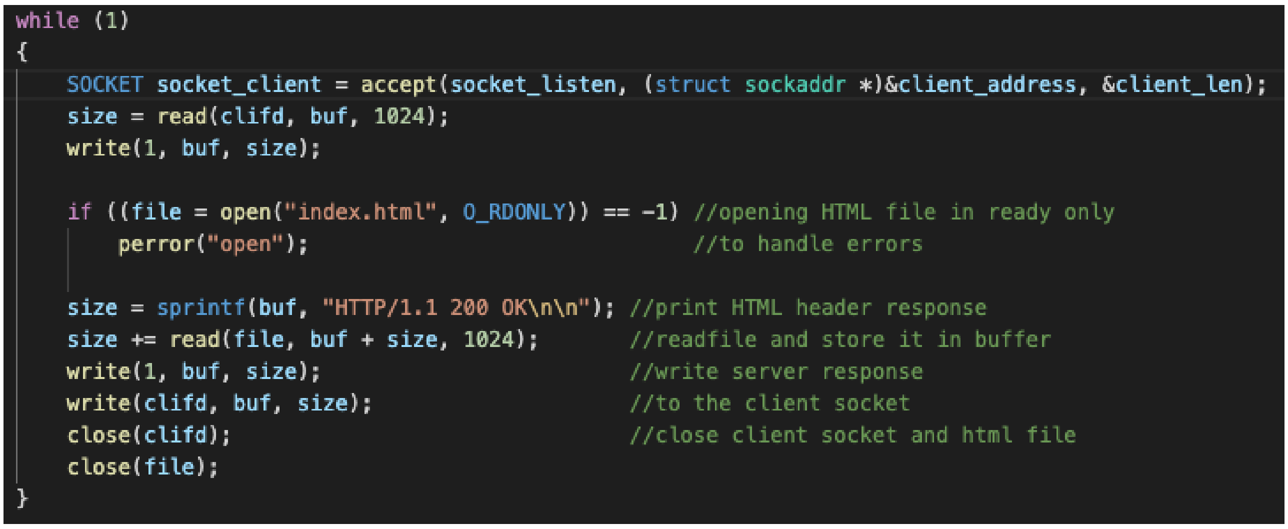
```c
if (listen(socket_listen, 10) < 0) //10 is use so listen() knows how many connection are allowed to queue up
{
    fprintf(stderr, "listen() failed. (%d)\n", GETSOCKETERRNO()); //error when listen() returns a value
    return 1;
}
```

We use this source code to manage the error of socket listening. Again, we use the MACRO defined at the begging to exit the program and return a different error message if socket listening fails.

III. Connection test

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.

Then we surround the accept() function with a while loop, so the server doesn’t quit automatically with this loop:



```c
while (1)
{
    SOCKET socket_client = accept(socket_listen, (struct sockaddr *)&client_address, &client_len);
    size = read(clifd, buf, 1024);
    write(1, buf, size);

    if ((file = open("index.html", O_RDONLY)) == -1) //opening HTML file in ready only
        perror("open");                             //to handle errors

    size = sprintf(buf, "HTTP/1.1 200 OK\n\n"); //print HTML header response
    size += read(file, buf + size, 1024);       //readfile and store it in buffer
    write(1, buf, size);                        //write server response
    write(clifd, buf, size);                    //to the client socket
    close(clifd);                               //close client socket and html file
    close(file);
}
```

Now we add this loop and the HTML file display in the console when everything works correctly and there is no bind error.