



**FAKULTI TEKNOLOGI KEJURUTERAAN  
ELEKTRIK DAN ELEKTRONIK  
UNIVERSITI TEKNIKAL MALAYSIA MELAKA**

<b>OPERATING SYSTEMS</b>		
<b>BEEC3453</b>	<b>SEMESTER 1</b>	<b>SESI 2021/2022</b>
<b>LAB 5: SOCKET PROGRAMMING</b>		
<b>NO.</b>	<b>STUDENTS' NAME</b>	<b>MATRIC. NO.</b>
1.	NOOR SHAFINA BINTI ABDUL GHANI	B081910293
2.	NORISA SHAFIKA BINTI MOHD GHANI	B081910039
3.	NUR ALYSHA BINTI NORAZIZAN	B081910155
4.	RAHMAN KAZI ASHIKUR	B081910450
<b>PROGRAMME</b>	3BEEC	
<b>SECTION GROUP</b>	S1/1	
<b>DATE</b>	21/12/2021	
<b>NAME OF INSTRUCTOR(S)</b>	1. NOOR MOHD ARIFF BIN BRAHIN	
	2.	
<b>EXAMINER'S COMMENT(S)</b>		<b>TOTAL MARKS</b>

Rev. No.	Date	Author(s)	Description
1.0	30 JAN 2019	1. Shamsul Fakhar 2. Noor Mohd Ariff	1. Update to new UTeM logo 2. Update faculty's name 3. Change "course" to "programme" 4. Remove verification stamp

## 1. LEARNING OUTCOMES

1. Differentiate the functionality among various kinds of OS components.
2. Manipulate OS theories to solve basic functional problems.
3. Perform lab and present technical report in writing.

## 2. REQUIREMENTS

1. PC with Linux Ubuntu installed (or any other POSIX-compliant system)
2. Knowledge of C programming language

## 3. SYNOPSIS & THEORY

Most inter-process communication uses the **client server model**. These terms refer to the two processes which will be communicating with each other. One of the two processes, the **client**, connects to the other process, the **server**, typically to make a **request for information**. A good analogy is a person who makes a phone call to another person.

Note that the **client** needs to know of the **existence** of and the **address** of the **server**, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established. Also note that once a **connection is established**, both sides can **send and receive information**.

The system calls for establishing a connection are somewhat different for the client and the server, but both involve the **basic construct of a socket**. A socket is one end of an inter-process communication channel. The two processes each establish their own socket.

The **steps** involved in establishing a **socket on the client side** are as follows:

- **Create** a socket with the **socket ()** system call
- **Connect** the socket to the address of the **server** using the **connect ()** system call
- **Send and receive data**. There are a number of ways to do this, but the simplest is to use the **send ()** and **recv ()** system calls

The **steps** involved in establishing a **socket on the server side** are as follows:

- **Create** a socket with the **socket ()** system call
- **Bind** the socket to an address using the **bind ()** system call. For a server socket on the internet, an address consists of a port number on the host machine
- **Listen for connections** with the **listen ()** system call
- **Accept a connection** with the **accept ()** system call. This call typically blocks until a client connects with the server
- **Send and receive data** with **send ()** and **recv ()** system calls

## 4. PROCEDURES

### PART 1: The Server

1. In terminal open a **text editor**.
2. Save the file as "server.c".
3. Write the code below into "**server.c**":

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>

#define RED "\x1B[31m"
#define GREEN "\x1B[32m"
#define YELLOW "\x1B[33m"
#define BLUE "\x1B[34m"
#define MAGENTA "\x1B[35m"
#define CYAN "\x1B[36m"
#define WHITE "\x1B[37m"
#define RESET "\033[0m"

int main()
{
    system("clear");

    /*
    printf(RED "red\t" RESET);
    printf(GREEN "green\t" RESET);
    printf(YELLOW "yellow\t" RESET);
    printf(BLUE "blue\t" RESET);
    printf(MAGENTA "magenta\t" RESET);
    printf(CYAN "cyan\t" RESET);
    printf(WHITE "white\t\n" RESET);
    printf("This is " RED "red" RESET " and this is "
    BLUE "blue" RESET "\n");
    */

    int sock;
    int connected;
    int receivedBytes;
    int true = 1;
    char outgoing [1024];
    char incoming[1024];
```

```

    struct sockaddr_in server_addr;
    struct sockaddr_in client_addr;
    int sin_size;

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        perror("Socket Error");
        exit(1);
    }

    if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR,
&true, sizeof(int)) == -1)
    {
        perror("Setsockopt Error");
        exit(1);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5000);
    server_addr.sin_addr.s_addr = INADDR_ANY;
    bzero(&(server_addr.sin_zero), 8);

    if (bind(sock, (struct sockaddr *)&server_addr,
sizeof(struct sockaddr)) == -1)
    {
        perror("Bind Error");
        exit(1);
    }

    if (listen(sock, 5) == -1)
    {
        perror("Listen Error");
        exit(1);
    }

    printf(YELLOW "TCP server is now up and
running...\n" RESET);
    printf(YELLOW "Now please run the client
program!\n" RESET);
    fflush(stdout);

    sin_size = sizeof(struct sockaddr_in);
    connected = accept(sock, (struct sockaddr
*)&client_addr, &sin_size);

    printf(YELLOW "Connection received from %s on port
%d\n" RESET, inet_ntoa(client_addr.sin_addr),
ntohs(client_addr.sin_port));
    printf(YELLOW "Chat server & client connection
successful.\n\n" RESET);

    printf("***** CHAT PROGRAM
(SERVER) *****\n");

```

```

while(1)
{
    printf(BLUE "SERVER: " RESET);
    fgets(outgoing, sizeof(outgoing), stdin);
    send(outgoing, outgoing, strlen(outgoing), 0);

    receivedBytes = recv(outgoing, incoming, 1024,
0);
    incoming[receivedBytes] = '\0';
    printf(GREEN "CLIENT: %s" RESET, incoming);
    fflush(stdout);
}
close(sock);
return 0;
}

```

4. **Compile** the code and make sure there's **no error**.
5. Obviously, it wouldn't make any sense to run the server program without a client program, so move on to Part 2.

## **PART 2: The Client**

1. Write the code below into a new file named "**client.c**":

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>

#define RED "\x1B[31m"
#define GREEN "\x1B[32m"
#define YELLOW "\x1B[33m"
#define BLUE "\x1B[34m"
#define MAGENTA "\x1B[35m"
#define CYAN "\x1B[36m"
#define WHITE "\x1B[37m"
#define RESET "\033[0m"

int main()
{
    system("clear");

    int sock;
    int receivedBytes;
    char outgoing[1024];
    char incoming[1024];
    struct hostent *host;
    struct sockaddr_in server_addr;

    host = gethostbyname("127.0.0.1");

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        perror("Socket");
        exit(1);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5000);
    server_addr.sin_addr = *((struct in_addr *)host->h_addr);
    bzero(&(server_addr.sin_zero), 8);

    if (connect(sock, (struct sockaddr *)&server_addr,
sizeof(struct sockaddr)) == -1)
    {
        perror("Connect");
        exit(1);
    }
}
```

```

    }

    printf("***** CHAT PROGRAM
(CLIENT) *****\n");

    while(1)
    {

        receivedBytes=recv(sock,incoming,1024,0);
        incoming[receivedBytes] = '\0';
        printf(BLUE "SERVER: %s" RESET, incoming);

        printf(GREEN "CLIENT: " RESET);
        fgets(outgoing, sizeof(outgoing), stdin);
        send(sock,outgoing,strlen(outgoing), 0);
    }

    return 0;
}

```

2. **Compile** the program.
3. Now that we have the server and the client, we can try to run these program to test it.
4. **Run** the **server** program.
5. Open **another Terminal** window, and **run** the **client** program here.
6. If everything is working, you can now exchange text between the server and client:

```

shamsul@shamsul-HP-Compaq-dc5850-Microtower: ~/Desktop
shamsul@shamsul-HP-Compaq-dc5850-Microtower:~/Desktop$ cd Desktop/
shamsul@shamsul-HP-Compaq-dc5850-Microtower:~/Desktop$ ./server

TCP server is now up and running...
Now please run the client program!
Connection received from 127.0.0.1 on port 34039
Chat server & client connection successful.

***** CHAT PROGRAM (SERVER) *****
SERVER: hai
CLIENT: hello
SERVER: nak berkenalan boleh?
CLIENT: huh, sape ni???
SERVER: 

```

```

shamsul@shamsul-HP-Compaq-dc5850-Microtower: ~/Desktop
***** CHAT PROGRAM (CLIENT) *****
SERVER: hai
CLIENT: hello
SERVER: nak berkenalan boleh?
CLIENT: huh, sape ni???

```



```
norisa@norisa-Virtua...~/Desktop/lab5_norisa - + x
File Edit Tabs Help
TCP server is now up and running...
Now please run the client program!
Connection received from 127.0.0.1 on port 57104
Chat server & client connection successful.

***** CHAT PROGRAM (SERVER) *****
*****

SERVER: hai
CLIENT: hello
SERVER: suka makan apa?
CLIENT: takoyaki :)
SERVER: 

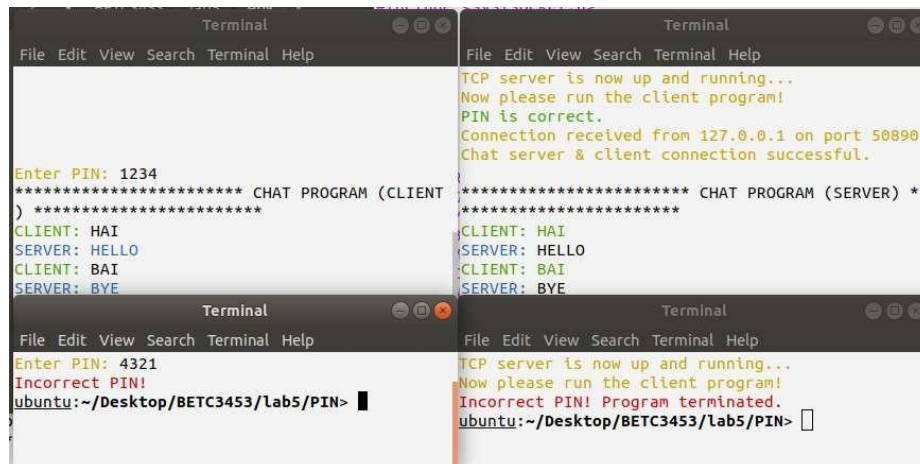
norisa@norisa-Virtua...~/Desktop/lab5_norisa - + x
File Edit Tabs Help
***** CHAT PROGRAM (CLIENT) *****
*****

SERVER: hai
CLIENT: hello
SERVER: suka makan apa?
CLIENT: takoyaki :)

```

### PART 3: Complete the Tasks:

1. Modify the client-server program according to the new specifications below:
  - a. **Client will be asked for a valid PIN number.** Whatever the user entered here will be **sent to server** for verification.  
(Note: **User will need to run server before running client program**)
  - b. **Server receives PIN** number from client and **checks whether it is correct.** There can be 2 possible outcomes of this process:
    - i. If PIN number is **not "1234"**, server program will **terminate** with message **"Incorrect PIN! Program terminated"**.  
At the same time, an **error message is sent to client**, and **client will also terminate** with message **"Incorrect PIN!"**
    - ii. If PIN number is **"1234"**, server will **display message "PIN is correct"** and **program will proceed.**
  - c. Chat program will run normally with client and server exchanging messages one message at a time. Note that **client will start sending the first message.**



```
Terminal
File Edit View Search Terminal Help

Enter PIN: 1234
***** CHAT PROGRAM (CLIENT) *****
CLIENT: HAI
SERVER: HELLO
CLIENT: BAI
SERVER: BYE

Terminal
File Edit View Search Terminal Help

TCP server is now up and running...
Now please run the client program!
PIN is correct.
Connection received from 127.0.0.1 on port 50890
Chat server & client connection successful.
***** CHAT PROGRAM (SERVER) *
*****
CLIENT: HAI
SERVER: HELLO
CLIENT: BAI
SERVER: BYE

Terminal
File Edit View Search Terminal Help

Enter PIN: 4321
Incorrect PIN!
ubuntu:~/Desktop/BETC3453/Lab5/PIN>

Terminal
File Edit View Search Terminal Help

TCP server is now up and running...
Now please run the client program!
Incorrect PIN! Program terminated.
ubuntu:~/Desktop/BETC3453/Lab5/PIN>
```

#### Hint:

```
//client
int main()
{
    socket()
    connect()

    // send username to server for verification
    send()

    // listen for message from server
    rcv()

    //check message from server
    if incoming=="ERROR"
        exit(1);

    //chat program starts
    while ()
}

//server
int main()
{
    socket()
    bind()
    listen()
    accept()

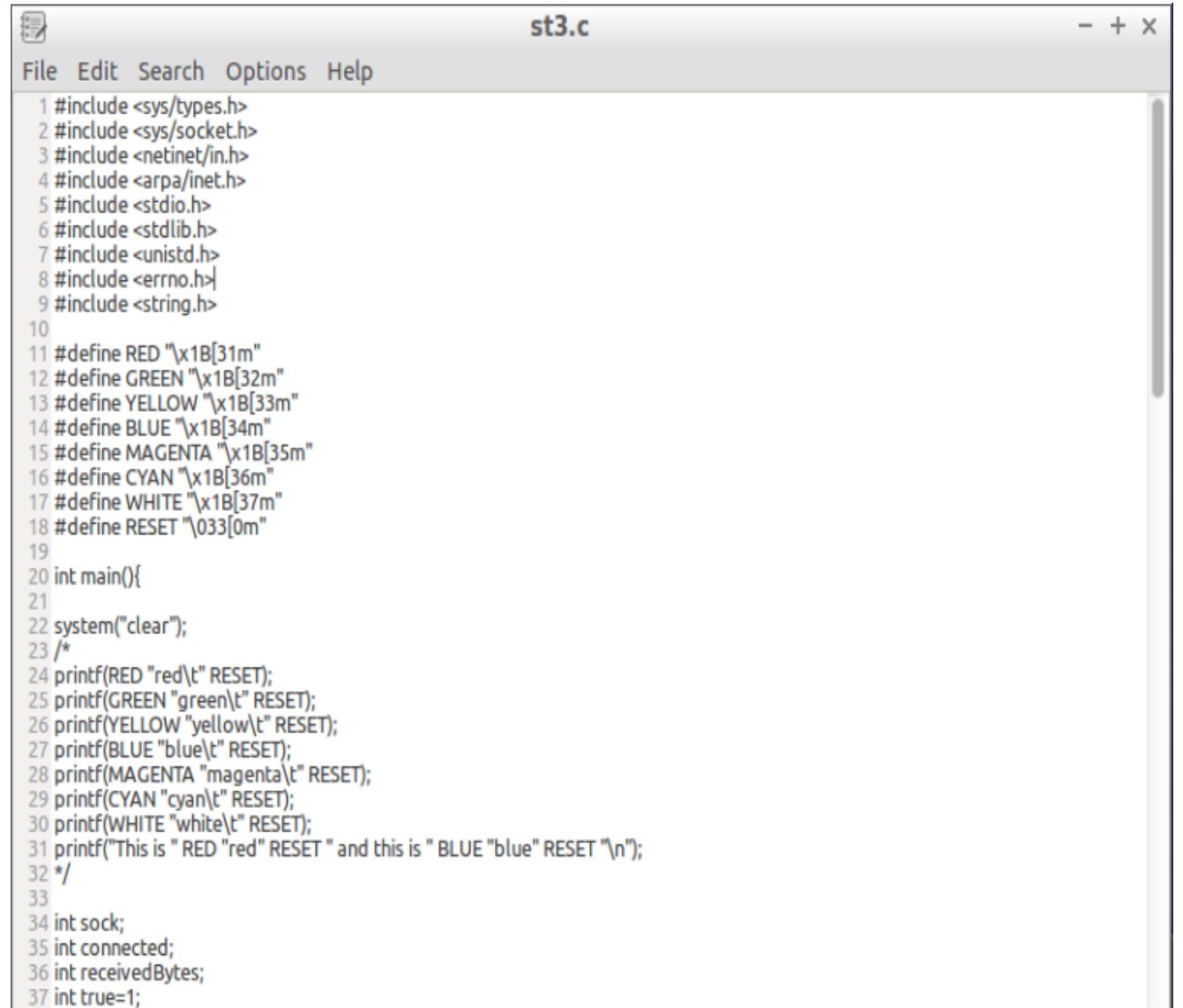
    // listen for PIN
    rcv()

    // check PIN
    if incoming != "1234" {
        send() //set outgoing = "ERROR"
        exit(1);
    }

    //chat program starts
    while ()
}
```

## CODING:

Server:



```
1 #include <sys/types.h>
2 #include <sys/socket.h>
3 #include <netinet/in.h>
4 #include <arpa/inet.h>
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <unistd.h>
8 #include <errno.h>
9 #include <string.h>
10
11 #define RED "\x1B[31m"
12 #define GREEN "\x1B[32m"
13 #define YELLOW "\x1B[33m"
14 #define BLUE "\x1B[34m"
15 #define MAGENTA "\x1B[35m"
16 #define CYAN "\x1B[36m"
17 #define WHITE "\x1B[37m"
18 #define RESET "\033[0m"
19
20 int main(){
21
22     system("clear");
23     /*
24     printf(RED "red\t" RESET);
25     printf(GREEN "green\t" RESET);
26     printf(YELLOW "yellow\t" RESET);
27     printf(BLUE "blue\t" RESET);
28     printf(MAGENTA "magenta\t" RESET);
29     printf(CYAN "cyan\t" RESET);
30     printf(WHITE "white\t" RESET);
31     printf("This is " RED "red" RESET " and this is " BLUE "blue" RESET "\n");
32     */
33
34     int sock;
35     int connected;
36     int receivedBytes;
37     int true=1;
```

```

38 char outgoing[1024];
39 char incoming[1024];
40 char pin[1024];
41 char pinNo[1024];
42 struct sockaddr_in server_addr;
43 struct sockaddr_in client_addr;
44 int sin_size;
45
46 if((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1){
47 perror("Socket Error");
48 exit(1);
49 }
50
51 if(setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &true, sizeof(int)) == -1){
52 perror("Setsockopt Error");
53 exit(1);
54 }
55
56 server_addr.sin_family=AF_INET;
57 server_addr.sin_port=htons(5000);
58 server_addr.sin_addr.s_addr=INADDR_ANY;
59 bzero(&(server_addr.sin_zero),8);
60
61 if(bind(sock, (struct sockaddr *) &server_addr, sizeof (struct sockaddr)) == -1){
62 perror("Bind Error");
63 exit(1);
64 }
65
66 if(listen(sock, 5) == -1){
67 perror("Listen Error");
68 exit(1);
69 }
70
71 printf(YELLOW "TCP server is now up and running...\n" RESET);
72 printf(YELLOW "Now please run the client program!\n" RESET);
73 fflush(stdout);
74

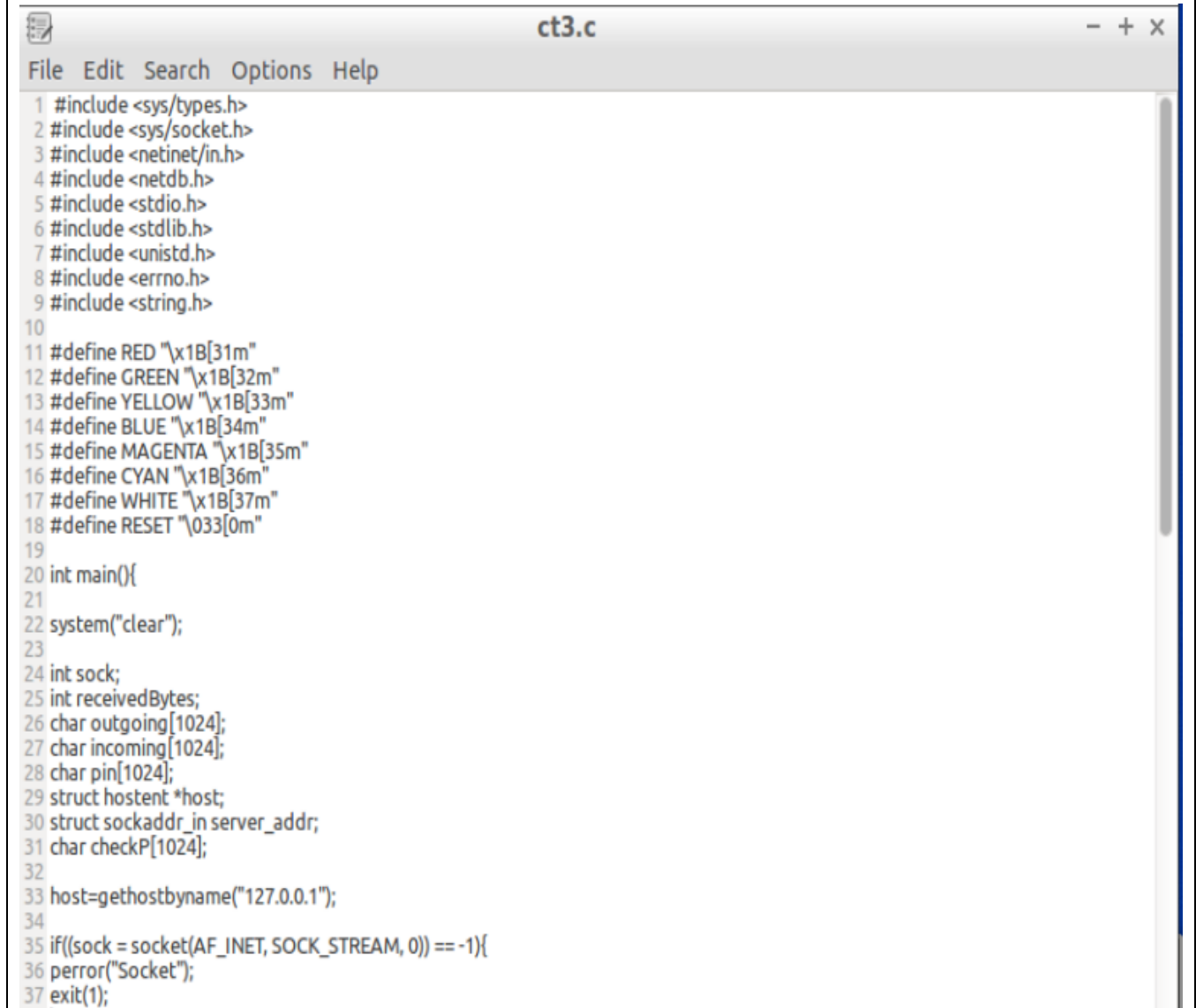
```

```

75 sin_size=sizeof(struct sockaddr_in);
76 connected=accept(sock, (struct sockaddr *) &client_addr, &sin_size);
77
78 //listen for pin
79 receivedBytes=recv(connected, pin, 1024, 0);
80 pin[receivedBytes]='\0';
81 fflush(stdout);
82 //give value to pinNO
83 strcpy(pinNo, "1234\n");
84
85 //check pin
86 if(strcmp(pin,pinNo)!=0){
87 printf(RED "Incorrect PIN! Program terminated.\n" RESET);
88 send(connected,"Incorrect_PIN!\n", 20,0);
89 exit(1);
90 }
91 else{
92 printf(GREEN "Pin is correct.\n" RESET);
93 send(connected,"Correct_PIN!\n", 14,0);
94 }
95 printf(YELLOW "Connection received from %s on port %d\n" RESET, inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
96 printf(YELLOW "Chat server & client connection successful. \n\n" RESET);
97 printf( "***** CHAT PROGRAM (SERVER) ***** \n\n");
98
99 while(1){
100 receivedBytes=recv(connected, incoming, 1024, 0);
101 incoming[receivedBytes]='\0';
102 printf(GREEN "CLIENT: %s" RESET, incoming);
103 fflush(stdout);
104
105 printf(BLUE "SERVER: " RESET);
106 fgets(outgoing, sizeof(outgoing), stdin);
107 send(connected, outgoing, strlen(outgoing), 0);
108
109 }
110 close(sock);
111 return 0;
112 }

```

Client:



```
1 #include <sys/types.h>
2 #include <sys/socket.h>
3 #include <netinet/in.h>
4 #include <netdb.h>
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <unistd.h>
8 #include <errno.h>
9 #include <string.h>
10
11 #define RED "\x1B[31m"
12 #define GREEN "\x1B[32m"
13 #define YELLOW "\x1B[33m"
14 #define BLUE "\x1B[34m"
15 #define MAGENTA "\x1B[35m"
16 #define CYAN "\x1B[36m"
17 #define WHITE "\x1B[37m"
18 #define RESET "\033[0m"
19
20 int main(){
21
22     system("clear");
23
24     int sock;
25     int receivedBytes;
26     char outgoing[1024];
27     char incoming[1024];
28     char pin[1024];
29     struct hostent *host;
30     struct sockaddr_in server_addr;
31     char checkP[1024];
32
33     host=gethostbyname("127.0.0.1");
34
35     if((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1){
36         perror("Socket");
37         exit(1);
38     }
```

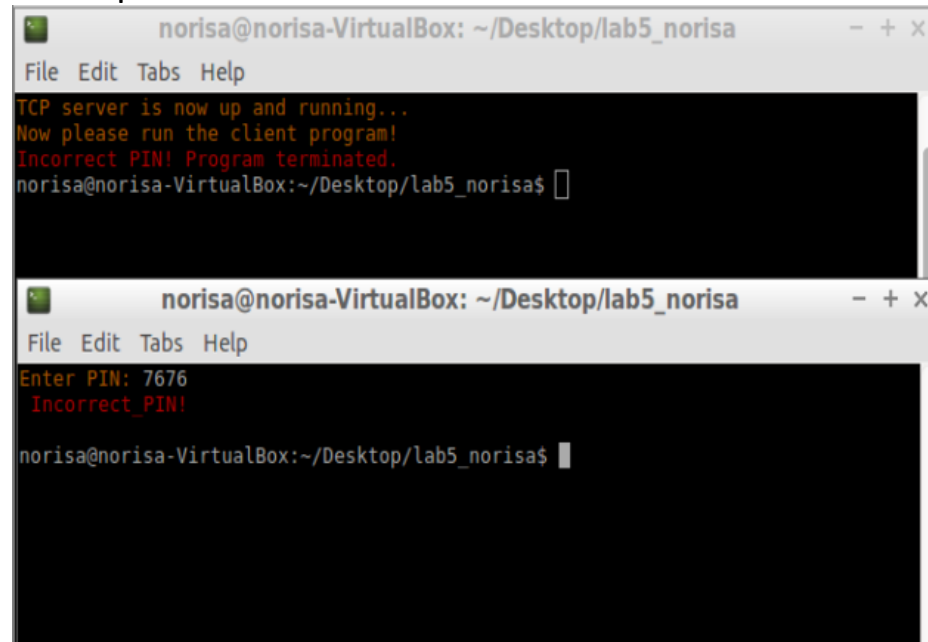
```

38 }
39
40 server_addr.sin_family=AF_INET;
41 server_addr.sin_port=htons(5000);
42 server_addr.sin_addr=((struct in_addr *)host->h_addr);
43 bzero(&(server_addr.sin_zero),8);
44
45 if(connect(sock, (struct sockaddr *) &server_addr, sizeof (struct sockaddr)) ==-1){
46 perror("Connect");
47 exit(1);
48 }
49
50 //send verification
51 printf(YELLOW "Enter PIN: " RESET);
52 fgets(pin, sizeof(pin), stdin);
53 send(sock, pin, strlen(pin), 0);
54
55 //check message
56 strcpy(checkP, "Incorrect_PIN!\n");
57 receivedBytes=recv(sock, incoming, 1024, 0);
58 incoming[receivedBytes]= '\0';
59
60 if(strcmp(incoming, checkP)==0){
61 printf(RED " %s\n" RESET, incoming);
62 exit(1);
63 }
64
65 printf( "***** CHAT PROGRAM (CLIENT) ***** \n");
66
67 while(1){
68 printf(GREEN "CLIENT: " RESET);
69 fgets(outgoing, sizeof(outgoing), stdin);
70 send(sock, outgoing, strlen(outgoing), 0);
71
72 receivedBytes=recv(sock, incoming, 1024, 0);
73 incoming[receivedBytes]= '\0';
74 printf(BLUE "SERVER: %s" RESET, incoming);
75
76 }
77 return 0;
78 }

```

## OUTPUT:

### Incorrect pin:

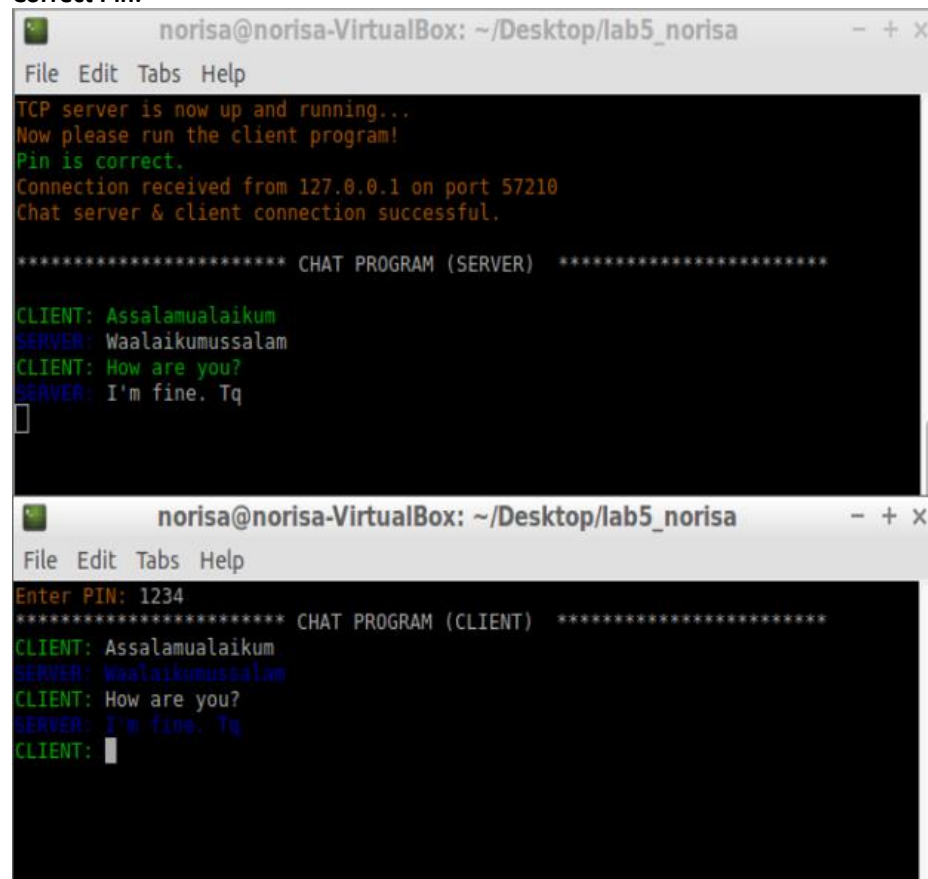


The first terminal window shows the server output: "TCP server is now up and running... Now please run the client program! Incorrect PIN! Program terminated." followed by the prompt "norisa@norisa-VirtualBox: ~/Desktop/lab5\_norisa\$". The second terminal window shows the client input "Enter PIN: 7676" and the server response "Incorrect PIN!" followed by the prompt "norisa@norisa-VirtualBox: ~/Desktop/lab5\_norisa\$".

```
norisa@norisa-VirtualBox: ~/Desktop/lab5_norisa
File Edit Tabs Help
TCP server is now up and running...
Now please run the client program!
Incorrect PIN! Program terminated.
norisa@norisa-VirtualBox:~/Desktop/lab5_norisa$

norisa@norisa-VirtualBox: ~/Desktop/lab5_norisa
File Edit Tabs Help
Enter PIN: 7676
Incorrect PIN!
norisa@norisa-VirtualBox:~/Desktop/lab5_norisa$
```

### Correct Pin:



The first terminal window shows the server output: "TCP server is now up and running... Now please run the client program! Pin is correct. Connection received from 127.0.0.1 on port 57210 Chat server & client connection successful." followed by a separator line "\*\*\*\*\* CHAT PROGRAM (SERVER) \*\*\*\*\*". The chat log shows: "CLIENT: Assalamualaikum", "SERVER: Waalaikumussalam", "CLIENT: How are you?", "SERVER: I'm fine. Tq". The second terminal window shows the client input "Enter PIN: 1234" followed by a separator line "\*\*\*\*\* CHAT PROGRAM (CLIENT) \*\*\*\*\*". The chat log shows: "CLIENT: Assalamualaikum", "SERVER: Waalaikumussalam", "CLIENT: How are you?", "SERVER: I'm fine. Tq", "CLIENT:".

```
norisa@norisa-VirtualBox: ~/Desktop/lab5_norisa
File Edit Tabs Help
TCP server is now up and running...
Now please run the client program!
Pin is correct.
Connection received from 127.0.0.1 on port 57210
Chat server & client connection successful.

***** CHAT PROGRAM (SERVER) *****

CLIENT: Assalamualaikum
SERVER: Waalaikumussalam
CLIENT: How are you?
SERVER: I'm fine. Tq

norisa@norisa-VirtualBox: ~/Desktop/lab5_norisa
File Edit Tabs Help
Enter PIN: 1234
***** CHAT PROGRAM (CLIENT) *****

CLIENT: Assalamualaikum
SERVER: Waalaikumussalam
CLIENT: How are you?
SERVER: I'm fine. Tq
CLIENT:
```



## 5. DISCUSSIONS & CONCLUSION

(What have you learned from this experiment?)

---

From this laboratory session, in part 1, we done coding to make the function of server where it will send message to the client first. So, we done the coding in part 2 for the client it will receive the message from the server and can reply to that message and both of them can keep receiving and sending message to each other. Then, for the task 3, we are required to do make a new coding for the server and client to send and receive each other, but need to input password first before doing that. So, the client will send the password first and then the server will check the password. The correct password is 1234, so if the client input password of 1234, the server will print Pin is Correct and they can start messaging where client will start to send message first to the server. But, if the client input different password other than 1234, they cannot start messaging and the message of "Incorrect PIN! Program terminated." Will be shown on server and message of "Incorrect PIN" will be shown on client side.

As a conclusion from this laboratory session, we are able to differentiate the functionality among various kinds of OS components. Then, we are also able to manipulate OS theories to solve basic functional problems and lastly, we are able to perform lab and present technical report in writing. Thus, objectives from this laboratory session are accomplished.