OPERATING SYSTEM LABORATORY MANUAL

UNIVERSITY OF THE PUNJAB

FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE

DEPARTMENT OF COMPUTER SCIENCE

Course:	**Operating System Lab**	**Date:**
Course Code:	**CC-217-3L**	**Max Marks: 40**
Faculty/Instructor's Name & Email:	**Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)**	

LAB MANUAL # 15

(SPRING 2023)

Name:_____ Enroll No: _____

Objective(s) :

To understand the concepts of socket programming with threads using the C programming language in a Linux environment. The lab includes an introduction, objectives, equipment/software requirements, code examples, and exercises.

Lab Tasks :

Task 1: What is Sockets Programming?
Task 2: Write and analyze code for 'client.c'
Task 3: Write and analyze code for 'server.c'
Task 4: Implement threads to support multiple client requests simultaneously.

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(*if any*)
1.	10		
2.	10		
3.	10		
4.	10		
Total	**40**		**Signature**

Note : Attempt all tasks and get them checked by your Instructor

Lab 15: Socket Programming using Threads in Linux

Objective(s):

To understand the concepts of socket programming using the C programming language in a Linux environment. The lab includes an introduction, objectives, equipment/software requirements, code examples, and exercises. This lab will introduce you to socket programming in a Linux environment using the C programming language. You will learn how to create a simple server that listens for incoming connections and a client that connects to the server. Additionally, you will implement threads to handle multiple client connections concurrently. These skills are fundamental for developing networked applications and understanding the basics of concurrent programming.

Socket Programming:

It is a fundamental aspect of network programming that allows processes to communicate over a network. This lab focuses on creating a simple server-client application using sockets in the Linux environment. Additionally, threads are employed to handle multiple client connections concurrently.

Implementations:

- Understand the basics of socket programming.
- Implement a simple server that listens for incoming connections.
- Develop a client program to connect to the server.
- Use threads to handle multiple client connections simultaneously.

Code Files:

- `'server.c'`: Code for the server application.
- `'client.c'`: Code for the client application.

Program Code for 'server.c'

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <pthread.h>

#define PORT 8080
#define MAX_CLIENTS 5

void *handleClient(void *socket_desc)
{
    int client_socket = *(int *)socket_desc;
    char buffer[1024] = {0};
    char *message = "Hello from server!\n";

    // Send a welcome message to the client
    send(client_socket, message, strlen(message), 0);

    // Receive data from the client
    recv(client_socket, buffer, sizeof(buffer), 0);
    printf("Client message: %s\n", buffer);

    // Close the socket and free the thread's resources
    close(client_socket);
    free(socket_desc);

    return NULL;
}

int main()
{
    int server_fd, client_socket;
    struct sockaddr_in server_addr, client_addr;
    pthread_t thread_id;

    // Create a socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
    {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
```

```
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(PORT);

// Bind the socket to the specified address and port
if (bind(server_fd, (struct sockaddr *)&server_addr,
    sizeof(server_addr)) < 0)
{
    perror("Bind failed");
    exit(EXIT_FAILURE);
}

// Listen for incoming connections
if (listen(server_fd, MAX_CLIENTS) < 0)
{
    perror("Listen failed");
    exit(EXIT_FAILURE);
}

printf("Server listening on port %d...\n", PORT);

while (1)
{
    int addr_len = sizeof(client_addr);

    // Accept a new connection
    if ((client_socket = accept(server_fd,
         (struct sockaddr *)&client_addr,
         (socklen_t *)&addr_len)) < 0)
    {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    // Create a new thread to handle the client
    int *new_socket = malloc(sizeof(int));
    *new_socket = client_socket;

    if (pthread_create(&thread_id,
           NULL, handleClient, (void *)new_socket) < 0)
    {
        perror("Thread creation failed");
        exit(EXIT_FAILURE);
    }
```

```
        // Detach the thread to allow it to run independently
        pthread_detach(thread_id);
    }

    return 0;
}
```

Program Code for 'client.c'

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080

int main()
{
    int sock = 0;
    struct sockaddr_in server_addr;
    char buffer[1024] = {0};

    // Create a socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0)
    {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }
```

```
    // Connect to the server
    if (connect(sock, (struct sockaddr *)&server_addr,
         sizeof(server_addr)) < 0)
    {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    // Receive a welcome message from the server
    recv(sock, buffer, sizeof(buffer), 0);
    printf("%s", buffer);

    // Send a message to the server
    char *message = "Hello from client!";
    send(sock, message, strlen(message), 0);

    close(sock);

    return 0;
}
```

Tasks and Exercises:

Compile and Run:

- Compile both server.c and client.c using GCC.
- Run the server and client in separate terminal windows.
- Observe the interaction between the client and server.

Multiple Clients:

- Modify the server code to handle multiple clients concurrently using threads.
- Test the server by connecting multiple clients simultaneously.

Enhancements:

- Add error handling to both the server and client code.
- Implement message exchange between the server and multiple clients.

Protocol Design:

- Design a simple protocol for communication between the server and clients.
- Modify the server and client code to adhere to the protocol.