MODULE 4

1. Explain the connection procedure followed in client server communication

Creating Sockets: Both the client and the server create their own sockets. A socket is a software endpoint that establishes communication between two hosts.

Binding (for servers): The server socket binds to a specific port on the host machine. This allows clients to know where to connect. Clients do not typically bind to a specific port; they usually connect to a server's port.

Connecting (for clients): The client socket initiates a connection request to the server's IP address and port number.

Listening (for servers): The server socket listens for incoming connection requests from clients. It waits for clients to connect to it.

Accepting Connections (for servers): When a client sends a connection request, the server socket accepts the request. This creates a new socket dedicated to that specific client, called the client socket.

Acceptance Confirmation (for clients): Once the server accepts the connection, the client socket receives confirmation that the connection is established.

Data Transfer: After the connection is established, data can be transmitted bidirectionally between the client and server through their respective sockets.

Closing Connection: Either the client or server (or both) can close the connection when the communication is complete. This involves closing the respective sockets.

2. What is the use of bind() function in socket programming ?

In socket programming, the bind() function is used to associate a socket with a specific network address (IP address and port number) on the local machine. This function is primarily used by servers to specify the port on which they will listen for incoming connections.

import socket

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_address = ('localhost', 8080)

server\_socket.bind(server\_address)

3. What is Datagram Socket ?

A datagram socket is a type of socket used in network communication that provides a connectionless, unreliable, packet-oriented communication. Datagram sockets are often associated with the User Datagram Protocol (UDP), although they can also be used with other protocols.

In programming, datagram sockets are typically created using functions like socket() in C or socket.socket() in Python, specifying the appropriate address family (such as AF\_INET for IPv4) and socket type (such as SOCK\_DGRAM).

4. Write a server/client model socket program to exchange hello message between them.

Server:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8888

#define BUFFER\_SIZE 1024

int main() {

int server\_socket, client\_socket;

struct sockaddr\_in server\_addr, client\_addr;

char buffer[BUFFER\_SIZE];

server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_socket == -1) {

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);

if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

if (listen(server\_socket, 1) == -1) {

perror("Listen failed");

exit(EXIT\_FAILURE);

}

printf("Server is listening...\n");

socklen\_t client\_addr\_len = sizeof(client\_addr);

client\_socket = accept(server\_socket, (struct sockaddr \*)&client\_addr, &client\_addr\_len);

if (client\_socket == -1) {

perror("Accept failed");

exit(EXIT\_FAILURE);

}

printf("Connection established with client\n");

ssize\_t bytes\_received = recv(client\_socket, buffer, BUFFER\_SIZE, 0);

if (bytes\_received == -1) {

perror("Receive failed");

exit(EXIT\_FAILURE);

}

printf("Received: %s\n", buffer);

char \*response = "Hello from server!";

if (send(client\_socket, response, strlen(response), 0) == -1) {

perror("Send failed");

exit(EXIT\_FAILURE);

}

close(client\_socket);

close(server\_socket);

return 0;

}

Client:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8888

#define BUFFER\_SIZE 1024

int main() {

int client\_socket;

struct sockaddr\_in server\_addr;

char buffer[BUFFER\_SIZE];

client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (client\_socket == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // Server IP address

server\_addr.sin\_port = htons(PORT);

if (connect(client\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Connection failed");

exit(EXIT\_FAILURE);

}

printf("Connected to server\n");

char \*message = "Hello from client!";

if (send(client\_socket, message, strlen(message), 0) == -1) {

perror("Send failed");

exit(EXIT\_FAILURE);

}

printf("Sent: %s\n", message);

ssize\_t bytes\_received = recv(client\_socket, buffer, BUFFER\_SIZE, 0);

if (bytes\_received == -1) {

perror("Receive failed");

exit(EXIT\_FAILURE);

}

printf("Received: %s\n", buffer);

close(client\_socket);

return 0;

}

5. Write a TCP server-client program to check if a given string is Palindrome

Input: level

Output: Palindrome

Input: Assessment

Output: Not a Palindrome

Server:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8888

#define BUFFER\_SIZE 1024

int isPalindrome(char \*str) {

int len = strlen(str);

for (int i = 0; i < len / 2; i++) {

if (str[i] != str[len - i - 1]) {

return 0; // Not a palindrome

}

}

return 1; // Palindrome

}

int main() {

int server\_socket, client\_socket;

struct sockaddr\_in server\_addr, client\_addr;

char buffer[BUFFER\_SIZE];

server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_socket == -1) {

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);

if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

if (listen(server\_socket, 1) == -1) {

perror("Listen failed");

exit(EXIT\_FAILURE);

}

printf("Server is listening...\n");

socklen\_t client\_addr\_len = sizeof(client\_addr);

client\_socket = accept(server\_socket, (struct sockaddr \*)&client\_addr, &client\_addr\_len);

if (client\_socket == -1) {

perror("Accept failed");

exit(EXIT\_FAILURE);

}

printf("Connection established with client\n");

ssize\_t bytes\_received = recv(client\_socket, buffer, BUFFER\_SIZE, 0);

if (bytes\_received == -1) {

perror("Receive failed");

exit(EXIT\_FAILURE);

}

buffer[bytes\_received] = '\0'; // Null-terminate the received data

if (isPalindrome(buffer)) {

send(client\_socket, "Palindrome", strlen("Palindrome"), 0);

} else {

send(client\_socket, "Not a Palindrome", strlen("Not a Palindrome"), 0);

}

close(client\_socket);

close(server\_socket);

return 0;

}

Client:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8888

#define BUFFER\_SIZE 1024

int main() {

int client\_socket;

struct sockaddr\_in server\_addr;

char buffer[BUFFER\_SIZE];

client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (client\_socket == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // Server IP address

server\_addr.sin\_port = htons(PORT);

if (connect(client\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Connection failed");

exit(EXIT\_FAILURE);

}

printf("Connected to server\n");

char \*message = "level";

if (send(client\_socket, message, strlen(message), 0) == -1) {

perror("Send failed");

exit(EXIT\_FAILURE);

}

printf("Sent: %s\n", message);

ssize\_t bytes\_received = recv(client\_socket, buffer, BUFFER\_SIZE, 0);

if (bytes\_received == -1) {

perror("Receive failed");

exit(EXIT\_FAILURE);

}

buffer[bytes\_received] = '\0'; // Null-terminate the received data

printf("Received: %s\n", buffer);

close(client\_socket);

return 0;

}

6. Write an example to demonstrate UDP server-client program

Server:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8888

#define BUFFER\_SIZE 1024

int main() {

int server\_socket;

struct sockaddr\_in server\_addr, client\_addr;

char buffer[BUFFER\_SIZE];

server\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (server\_socket == -1) {

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);

if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

printf("Server is listening...\n");

while (1) {

socklen\_t client\_addr\_len = sizeof(client\_addr);

ssize\_t bytes\_received = recvfrom(server\_socket, buffer, BUFFER\_SIZE, 0,

(struct sockaddr \*)&client\_addr, &client\_addr\_len);

if (bytes\_received == -1) {

perror("Receive failed");

exit(EXIT\_FAILURE);

}

buffer[bytes\_received] = '\0';

printf("Received from client: %s\n", buffer);

if (sendto(server\_socket, buffer, bytes\_received, 0,

(struct sockaddr \*)&client\_addr, client\_addr\_len) == -1) {

perror("Send failed");

exit(EXIT\_FAILURE);

}

}

close(server\_socket);

return 0;

}

Client:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8888

#define BUFFER\_SIZE 1024

int main() {

int client\_socket;

struct sockaddr\_in server\_addr;

char buffer[BUFFER\_SIZE];

client\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (client\_socket == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // Server IP address

server\_addr.sin\_port = htons(PORT);

printf("Enter message to send: ");

fgets(buffer, BUFFER\_SIZE, stdin);

if (sendto(client\_socket, buffer, strlen(buffer), 0,

(struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Send failed");

exit(EXIT\_FAILURE);

}

socklen\_t server\_addr\_len = sizeof(server\_addr);

ssize\_t bytes\_received = recvfrom(client\_socket, buffer, BUFFER\_SIZE, 0,

(struct sockaddr \*)&server\_addr, &server\_addr\_len);

if (bytes\_received == -1) {

perror("Receive failed");

exit(EXIT\_FAILURE);

}

buffer[bytes\_received] = '\0'; // Null-terminate the received data

printf("Received from server: %s\n", buffer);

close(client\_socket);

return 0;

}