Student name: MWALE RAMSY

Student ID: 22/05396

1) Explain the concept of client-server architecture.

The **Client-Server Architecture** is a network model where multiple clients request and receive services from a centralized server.
The **server** is a system that hosts resources and services, handling multiple client requests. The **client** initiates communication, requesting resources or services from the server.
Working of Client-Server Model:
The **server** starts and listens for client requests.
A **client** establishes a connection to the server.
The server **processes** the request and **sends a response**.
The connection is maintained for further communication or closed after processing.
Example Use Cases
Web servers (HTTP/HTTPS) (Clients: Web browsers)
Email servers (SMTP, POP3, IMAP)
Database servers (SQL, NoSQL)

2) Explain the different I/O models in Unix operating system.

UNIX supports multiple **I/O models** for handling input/output operations. The five main models are:
a. Blocking I/O
The process is **blocked** until the data is available.
Simple but inefficient for real-time applications.

b. Non-Blocking I/O
The process does **not block** but instead checks for data availability and returns immediately.
Uses system calls like fcntl(fd, F_SETFL, O_NONBLOCK).

c. I/O Multiplexing (select, poll, epoll)
Allows a process to monitor multiple file descriptors.
Functions: select(), poll(), epoll().

d. Signal-Driven I/O

Uses **signals** to notify the process when data is available.
The function sigaction() is used to register a signal handler.

e. Asynchronous I/O (AIO)

Uses system calls like aio_read() and aio_write() to perform non-blocking I/O. Efficient but complex to implement.

3) Define byte ordering function.

Byte ordering functions handle **endianness**, which determines how multi-byte data is stored in memory.

Types of Endianness

Big-endian: Most significant byte stored first.
Little-endian: Least significant byte stored first.
Byte Ordering Functions in C
Host to Network ○ htons(x): Converts 16-bit short from host to network byte order.
- ○ htonl(x): Converts 32-bit long from host to network byte order.

Network to Host ○ ntohs(x): Converts 16-bit short from network to host byte order. ○ ntohl(x): Converts 32-bit long from network to host byte order.

4) Discuss the socket, connect, bind, listen and accept functions.

1. socket()

Creates a socket for communication.

int sockfd = socket(AF_INET, SOCK_STREAM, 0);

AF_INET: IPv4
SOCK_STREAM: TCP socket

2. bind()

Associates the socket with an address and port.

bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));

3. listen()

Marks the socket as **passive**, meaning it can accept connections.

listen(sockfd, 5);

5 is the backlog (number of pending connections).

3. accept()

Accepts a connection request from a client.

int client_fd = accept(sockfd, (struct sockaddr*)&client_addr, &addrlen);

4. connect()

Used by a client to establish a connection to a server.

connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));

5) Explain in detail about address conversion functions.

1. inet_pton()

Converts an IP address from text ("192.168.1.1") to binary format

inet_pton(AF_INET, "192.168.1.1", &addr);

2. inet_ntop()

Converts an IP address from binary to text format.

```
char str[INET_ADDRSTRLEN];
inet_ntop(AF_INET, &addr, str, INET_ADDRSTRLEN);
```

6) Explain the concept of shutdown function.

The shutdown() function is used to **partially** or **fully** close a socket connection.

```
shutdown(sockfd, how);
```

how = 0 → Stop receiving. how
= 1 → Stop sending.
how = 2 → Stop both sending and receiving.

7) Explain how to handle server host crashes and shutdowns.

When a server crashes or shuts down unexpectedly, clients must handle disconnections gracefully.

Solutions Use TCP Keep-Alive

Detect inactive connections with periodic probes. Example:

```
int keepalive = 1;
setsockopt(sockfd, SOL_SOCKET, SO_KEEPALIVE, &keepalive, sizeof(keepalive));
```

Implement Timeout Handling Use setsockopt() to set a timeout for socket operations.

Use Reconnection Logic Attempt to reconnect if the connection is lost.

8) Write a program to implement an echo server/client using TCP sockets.

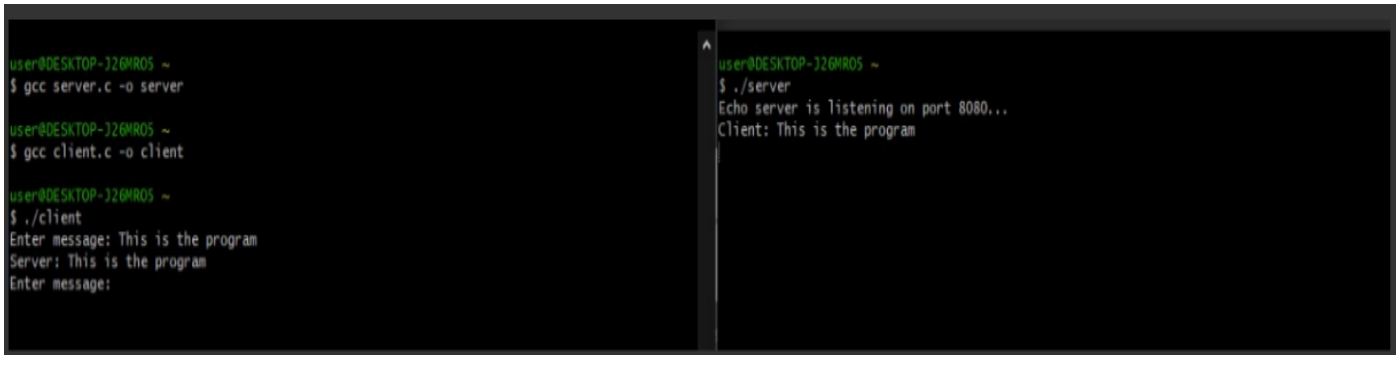
Echo Server (C)

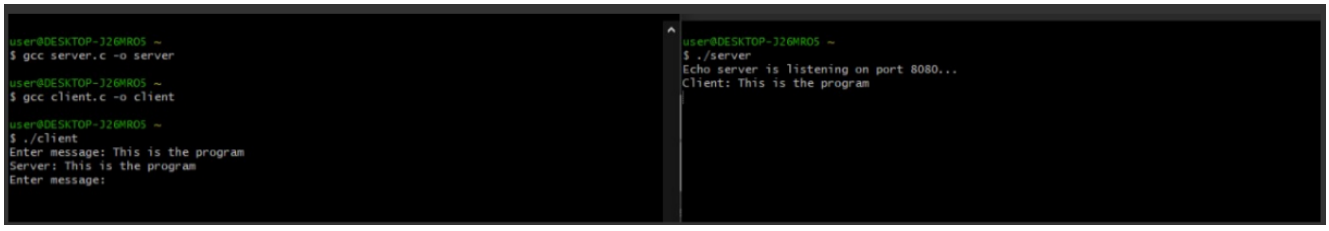
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

int main() {     int
sockfd, new_sock;
struct sockaddr_in
server_addr,
client_addr;     char
buffer[1024];
   socklen_t addr_size;

   sockfd = socket(AF_INET, SOCK_STREAM, 0);
server_addr.sin_family = AF_INET;     server_addr.sin_port
= htons(8080);
   server_addr.sin_addr.s_addr = INADDR_ANY;

   bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
listen(sockfd, 5);

   addr_size = sizeof(client_addr);     new_sock = accept(sockfd, (struct
sockaddr*)&client_addr, &addr_size);

   while (1) {         recv(new_sock,
buffer, 1024, 0);
      send(new_sock, buffer, strlen(buffer), 0);
   }

   close(sockfd);
return 0;
}
```

Echo Client (C)

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

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

int main() {     int sockfd;     struct
sockaddr_in server_addr;     char
buffer[1024];

   sockfd = socket(AF_INET, SOCK_STREAM, 0);
server_addr.sin_family = AF_INET;     server_addr.sin_port
= htons(8080);     inet_pton(AF_INET, "127.0.0.1",
&server_addr.sin_addr);

   connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));

   while (1) {
      printf("Enter message: ");
fgets(buffer, 1024, stdin);         send(sockfd,
buffer, strlen(buffer), 0);         recv(sockfd,
buffer, 1024, 0);
      printf("Server: %s", buffer);
   }

   close(sockfd);
return 0;
}
```





9) Write a program to implement a file transfer application.

File Transfer Server (C)

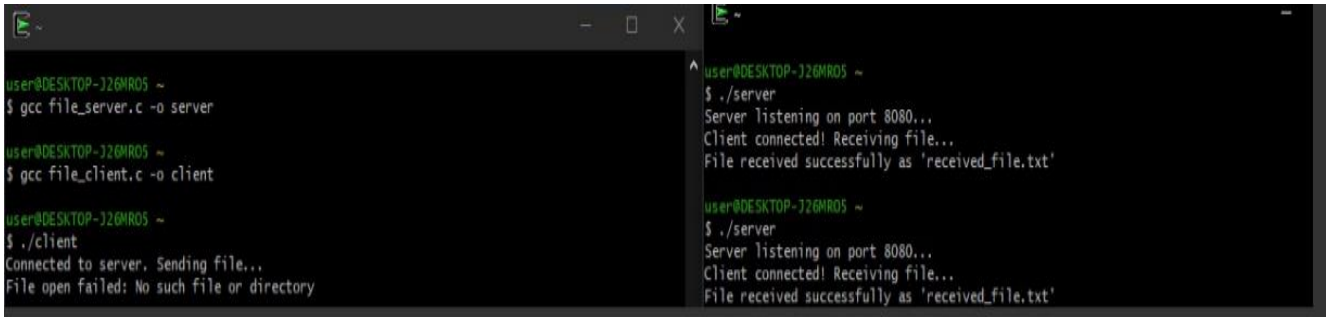
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define FILENAME "server_file.txt"

int main() {     int server_fd, client_fd;
struct sockaddr_in server_addr, client_addr;
char buffer[1024];     socklen_t addr_size;
FILE *file;

    // Create socket
    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = INADDR_ANY;

    // Bind socket     bind(server_fd, (struct sockaddr*)&server_addr,
sizeof(server_addr));

    // Listen for connections
listen(server_fd, 5);
    printf("Server listening on port %d...\n", PORT);
```

```
   // Accept client connection     addr_size = sizeof(client_addr);     client_fd
= accept(server_fd, (struct sockaddr*)&client_addr, &addr_size);
printf("Client connected.\n");

   // Open file for reading     file
= fopen(FILENAME, "r");     if
(file == NULL) {
perror("File not found");
close(server_fd);         return 1;
}

   // Send file contents     while (fgets(buffer,
sizeof(buffer), file) != NULL) {
       send(client_fd, buffer, strlen(buffer), 0);
   }

   printf("File sent successfully.\n");

   fclose(file);
close(client_fd);
close(server_fd);
return 0; }
```

File Transfer Client (C)

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

#define PORT 8080
#define FILENAME "received_file.txt"

int main() {     int sockfd;
struct sockaddr_in server_addr;
char buffer[1024];     FILE *file;

   // Create socket
```

```
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
server_addr.sin_family = AF_INET;     server_addr.sin_port
= htons(PORT);
    inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr);

    // Connect to server
    if (connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr)) == -1) {
perror("Connection failed");
        return 1;
    }

    printf("Connected to server. Receiving file...\n");

    // Open file for writing     file =
fopen(FILENAME, "w");     if
(file == NULL) {
perror("Error creating file");
        return 1;
    }

    // Receive file data
    while (recv(sockfd, buffer, sizeof(buffer), 0) > 0) {
fputs(buffer, file);
    }

    printf("File received successfully.\n");

    fclose(file);
close(sockfd);
return 0;
}
```


~
user@DESKTOP-J26MRO5 ~
$ gcc file_server.c -o server
user@DESKTOP-J26MRO5 ~
$ gcc file_client.c -o client
user@DESKTOP-J26MRO5 ~
$ ./client
Connected to server. Sending file...
File open failed: No such file or directory
~
user@DESKTOP-J26MRO5 ~
$ ./server
Server listening on port 8080...
Client connected! Receiving file...
File received successfully as 'received_file.txt'
user@DESKTOP-J26MRO5 ~
$ ./server
Server listening on port 8080...
Client connected! Receiving file...
File received successfully as 'received_file.txt'