**2.6)**
**Server File**

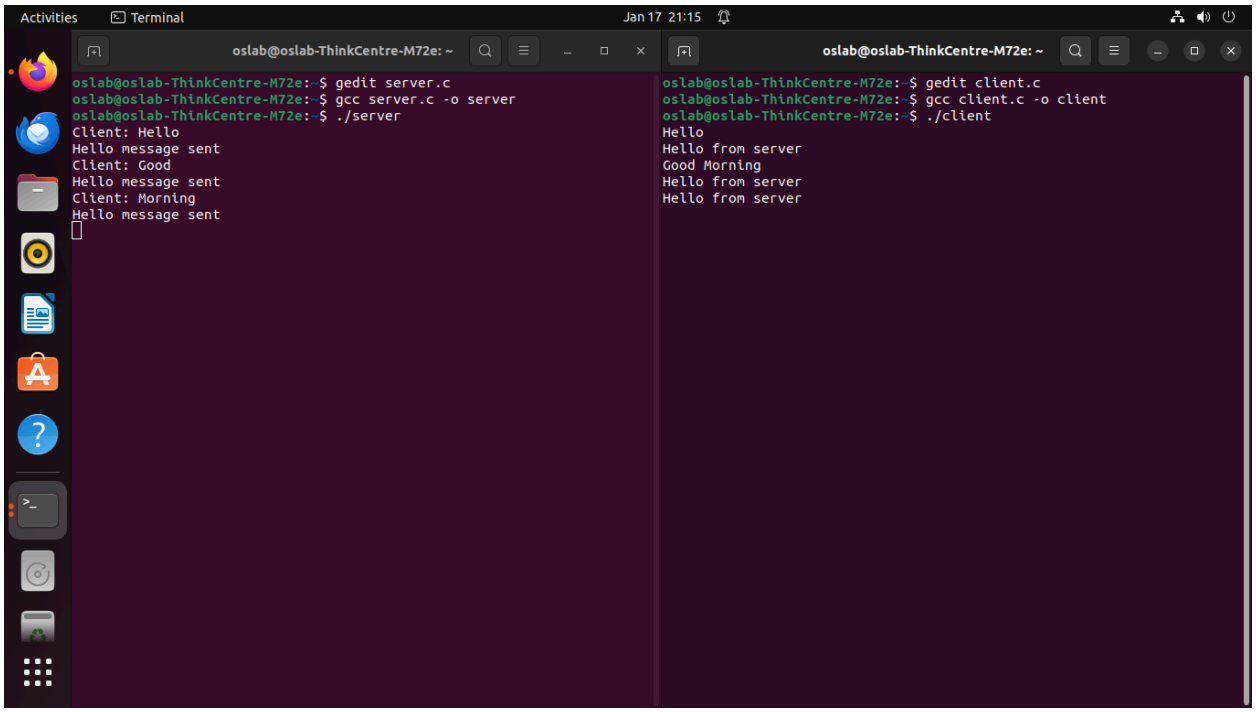
```
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#define PORT 8080
int main(int argc, char const *argv[]) {
int server_fd, new_socket, valread;
struct sockaddr_in address;
int opt = 1;
int addrlen = sizeof(address);
char buffer[1024];
char *hello = "Hello from server";
// Creating socket file descriptor
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
perror("socket failed");
exit(EXIT_FAILURE);
}
// Forcefully attaching socket to the port 8080 - For address reuse
if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt,
sizeof(opt))) {
perror("setsockopt");
exit(EXIT_FAILURE);
}
address.sin_family = AF_INET; // match the socket() call
address.sin_addr.s_addr = INADDR_ANY; // bind to any local address
address.sin_port = htons(PORT); // specify port to listen on
// Forcefully attaching socket to the port 8080
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
perror("bind failed");
exit(EXIT_FAILURE);
}
if (listen(server_fd, 3) < 0) {
perror("listen");
exit(EXIT_FAILURE);
}

if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) < 0) {
perror("accept");
exit(EXIT_FAILURE);
```

```
}
while (1) {
valread = read(new_socket, buffer, 1024);
printf("Client: %s\n", buffer);
memset(buffer, 0, 1024);
send(new_socket, hello, strlen(hello), 0);
printf("Hello message sent\n");
}
return 0;
}
```

**Client code:**

```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#define PORT 8080
int main(int argc, char const *argv[]) {
int sock = 0, valread;
struct sockaddr_in serv_addr;
char *exit_msg = "exit", *msg;
char buffer[1024] = {0};
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
printf("\nSocket creation error\n");
return -1;
}
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);
// Convert IPv4 and IPv6 addresses from text to binary form
if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
printf("\nInvalid address/Address not supported\n");
return -1;
}
if (connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0) {
printf("\nConnection Failed\n");
return -1;
}
while (1) {
scanf("%s", msg);
if (!strcmp(msg, exit_msg)) {
close(sock);
return 0;
}
```

```
send(sock, msg, strlen(msg), 0);
valread = read(sock, buffer, 1024);
printf("%s\n", buffer);
}
return 0;
}
```




**Question 3.3**

**Q-1)**
**Server file**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define PORT 8080

int main() {
        int server_fd, new_socket;
        struct sockaddr_in address;
        int opt = 1;
        int addrlen = sizeof(address);
```

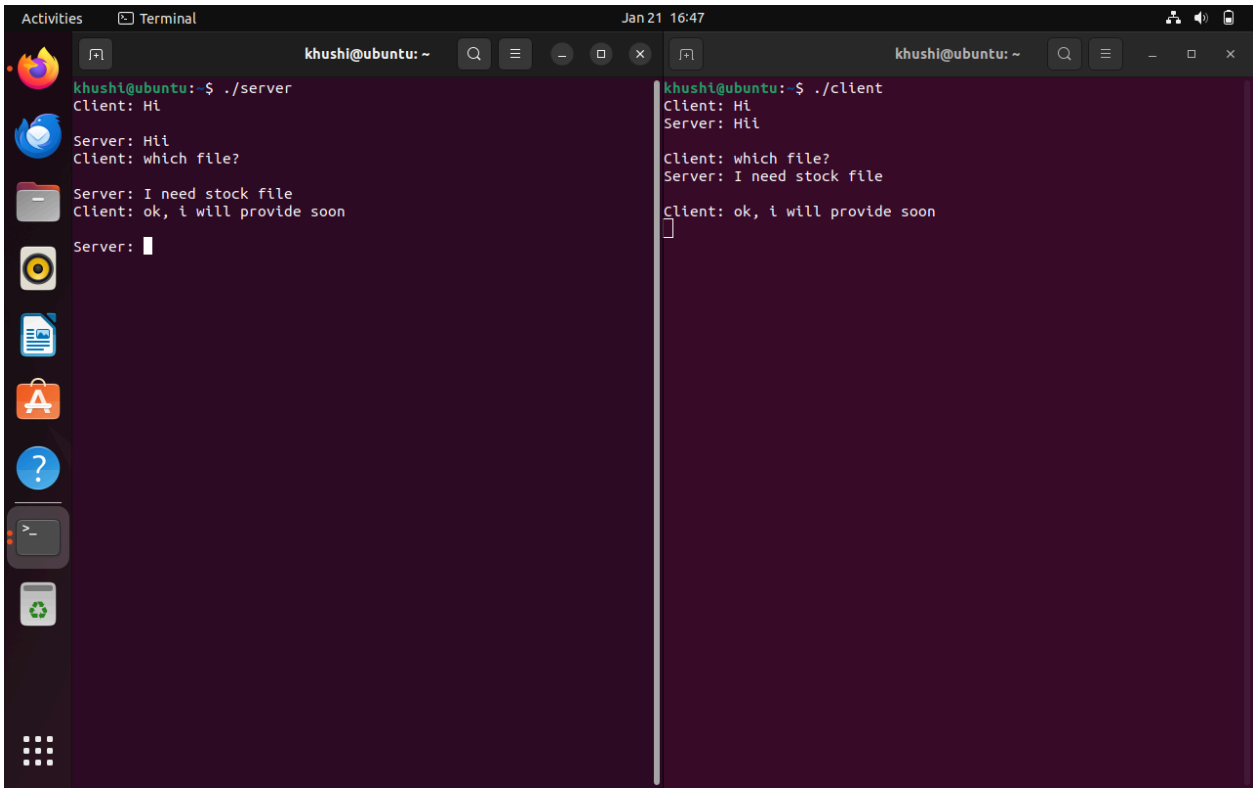
```
char buffer[1024] = {0};
char *quit_message = "QUIT";

// Create socket file descriptor
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
perror("socket failed");
exit(EXIT_FAILURE);
}

// Forcefully attach socket to the port
if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt,
sizeof(opt))) {
perror("setsockopt");
exit(EXIT_FAILURE);
}

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

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

// Listen for incoming connections
if (listen(server_fd, 3) < 0) {
perror("listen");
exit(EXIT_FAILURE);
}

if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen))
< 0) {
perror("accept");
exit(EXIT_FAILURE);
}

// Chat loop
while (1) {
int valread = read(new_socket, buffer, 1024);
if (valread < 0) {
perror("read");
break;
```

```
        }

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

        if (strcmp(buffer, quit_message) == 0) {
        printf("Client terminated the connection.\n");
        break;
        }

        printf("Server: ");
        fgets(buffer, 1024, stdin);
        // buffer[strcspn(buffer, "\n")] = '\0'; // Remove trailing newline
    fflush(stdin);
        send(new_socket, buffer, strlen(buffer), 0);

        if (strcmp(buffer, quit_message) == 0) {
        printf("Server terminated the connection.\n");
        break;
        }
        }

        // Close sockets
        close(new_socket);
        close(server_fd);
        return 0;
}
```

**Client File**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define PORT 8080

int main() {
        int sock = 0;
        struct sockaddr_in serv_addr;
        char buffer[1024] = {0};
        char *quit_message = "QUIT";
```

```
        if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1;
        }

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

        // Convert IPv4 and IPv6 addresses from text to binary form
        if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
        }

        if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        printf("\nConnection Failed \n");
        return -1;
        }

        // Chat loop
        while (1) {
        printf("Client: ");
        fgets(buffer, 1024, stdin);
        //buffer[strcspn(buffer, "\n")] = '\0'; // Remove trailing newline
    fflush(stdin);
        send(sock, buffer, strlen(buffer), 0);

        if (strcmp(buffer, quit_message) == 0) {
        printf("Client terminated the connection.\n");
        break;
        }

        int valread = read(sock, buffer, 1024);
        if (valread < 0) {
        perror("read");
        break;
        }

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

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




**Question 3.3**
**Q-2)**

**Server File**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <fcntl.h>
#define PORT 8080
#define SEGMENT_SIZE 50
#define BUFFER_SIZE 1500

typedef struct {
        int client_socket;
        char filename[256];
} client_data;

void *handle_client(void *arg) {
```

```
        client_data *data = (client_data *)arg;
        int client_socket = data->client_socket;
        char filename[256];
        strcpy(filename, data->filename);
        int file_descriptor = open(filename, O_WRONLY | O_CREAT, 0644);
        char buffer[BUFFER_SIZE];
        ssize_t received_bytes;
        size_t total_bytes = 0;

        while ((received_bytes = recv(client_socket, buffer, SEGMENT_SIZE, 0)) > 0) {
        write(file_descriptor, buffer, received_bytes);
        total_bytes += received_bytes;
        printf("Received segment of %d bytes\n", (int)received_bytes);
}


        // Check for "SEND-COMPLETE" after the loop
        if (strstr(buffer, "SEND-COMPLETE") != NULL) {
        printf("File %s received completely\n", filename);
        send(client_socket, "FILE-RECEIVED", strlen("FILE-RECEIVED") + 1, 0);
}


        // Close the file and socket
        close(file_descriptor);
        close(client_socket);
        free(data);

        return NULL;
}


int main() {
        int server_socket, client_socket;
        struct sockaddr_in server_addr, client_addr;
        socklen_t addr_len = sizeof(client_addr);

        // Buffer to store received data
        char buffer[BUFFER_SIZE];

        // Create a socket
        server_socket = socket(AF_INET, SOCK_STREAM, 0);

        // Configure the server address structure
```

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

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

    // Listen for incoming connections
    listen(server_socket, 3);

    printf("Server started and waiting for clients...\n");

    while ((client_socket = accept(server_socket, (struct sockaddr *)&client_addr,
&addr_len))) {
    printf("Client connected\n");

    // Receive filename from the client
    recv(client_socket, buffer, BUFFER_SIZE, 0);
    printf("Received filename: %s\n", buffer);

    // Create a data structure to pass to the thread
    client_data *data = malloc(sizeof(client_data));
    strcpy(data->filename, buffer);
    data->client_socket = client_socket;

    // Create a new thread to handle the client
    pthread_t thread_id;
    pthread_create(&thread_id, NULL, handle_client, (void *)data);
    pthread_detach(thread_id);
    }

    // Close the server socket
    close(server_socket);

    return 0;
}
```

**Client File**

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

```
#include <fcntl.h>
#include <sys/sendfile.h>

#define PORT 8080
#define SEGMENT_SIZE 50
#define BUFFER_SIZE 1500

int main(int argc, char *argv[]) {
        if (argc != 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
        }

        int client_socket, file_descriptor;
        struct sockaddr_in server_addr;

        client_socket = socket(AF_INET, SOCK_STREAM, 0);
        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
        server_addr.sin_port = htons(PORT);

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

   //size_t size = sizeof(argv[1]);
   char fname[256];
   printf("Enter filename : ");
   scanf("%s",fname);
        send(client_socket, fname, sizeof(fname)+1, 0);

        file_descriptor = open(argv[1], O_RDONLY);
        char buffer[BUFFER_SIZE];
        ssize_t sent_bytes;
        size_t total_bytes = 0;

        while ((sent_bytes = sendfile(client_socket, file_descriptor, NULL, SEGMENT_SIZE)) >
0) {
        total_bytes += sent_bytes;
        printf("Sent segment of %d bytes\n", (int)sent_bytes);
}


        send(client_socket, "SEND-COMPLETE", strlen("SEND-COMPLETE") + 1, 0);
        printf("SEND-COMPLETE");
```

```
        close(file_descriptor);
        close(client_socket);

        return 0;
}
```