NITHIN S
221IT085

Write a socket program, where the client sends content of file and server sends the number of characters in the file.

SERVER CODE **server.c**

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

int main()
{
   char recvContent[1024];
   int count;

   // Create a socket for the server
   int server_socket = socket(AF_INET, SOCK_STREAM, 0);
   if (server_socket == -1)
   {
      perror("Error creating server socket");
      exit(1);
   }
   printf("Waiting for connection...\n");

   // Initialize server address structure
   struct sockaddr_in servaddr;
   bzero(&servaddr, sizeof(servaddr));
   servaddr.sin_family = AF_INET;
   servaddr.sin_port = htons(8080);

   // Bind the socket to the server address
   if (bind(server_socket, (struct sockaddr *)&servaddr, sizeof(servaddr)) == -1)
   {
      perror("Error binding server socket");
      close(server_socket);
      exit(1);
   }

   // Listen for incoming connections
   if (listen(server_socket, 1) == -1)
```

```
{
    perror("Error listening for connections");
    close(server_socket);
    exit(1);
}

// Accept a client connection
int clifd = accept(server_socket, NULL, NULL);
if (clifd == -1)
{
    perror("Error accepting client connection");
    close(server_socket);
    exit(1);
}

printf("Client Connected.\n");

// Receive data from the client
ssize_t end = read(clifd, recvContent, sizeof(recvContent));
if (end == -1)
{
    perror("Error reading data from client");
    close(clifd);
    close(server_socket);
    exit(1);
}
recvContent[end] = '\0';

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

// Count the characters received
for (count = 0; recvContent[count] != '\0'; count++)
{
}

// Convert count to network byte order
int countToSend = htonl(count);

// Send the character count back to the client
if (send(clifd, &countToSend, sizeof(countToSend), 0) == -1)
{
    perror("Error writing data to client");
    close(clifd);
    close(server_socket);
    exit(1);
}

// Close the client and server sockets
close(clifd);
close(server_socket);

return 0;
```

}

This server code is a simple implementation of a TCP server using socket programming in C. It performs the following tasks:

1. **Socket Creation:**
   - It creates a socket using the `socket()` function with the address family `AF_INET` (IPv4), socket type `SOCK_STREAM` (for TCP), and protocol `0` (default protocol).

2. **Server Address Initialization:**
   - It initializes a `struct sockaddr_in` named `servaddr` to represent the server's address.
   - It sets the address family to `AF_INET`.
   - It specifies the port number using `htons()` to convert it to network byte order.

3. **Binding:**
   - It binds the socket to the server address using the `bind()` function. If binding fails, it handles the error and exits.

4. **Listening:**
   - It listens for incoming connections using the `listen()` function. If listening fails, it handles the error and exits.

5. **Accepting Connections:**
   - It accepts incoming client connections using the `accept()` function. If accepting a connection fails, it handles the error and exits.
   - Once a client connection is accepted, it returns a new socket descriptor (`clifd`) to communicate with the client.

6. **Data Reception:**
   - It receives data sent by the client using the `read()` function. The received data is stored in the `recvContent` buffer.
   - If there's an error while reading data, it handles the error and exits.

7. **Character Counting:**
   - It counts the number of characters received from the client by iterating through the `recvContent` buffer.

8. **Converting Count:**
   - It converts the character count to network byte order using `htonl()` to ensure consistent representation for transmission.

9. **Sending Data:**
   - It sends the character count back to the client using the `send()` function.

10. **Cleaning Up:**
    - It closes both the client and server sockets using `close()`.
    - The server code then returns `0` to indicate successful execution.

This server listens for incoming connections on port `8080`, receives a message from the client, counts the characters in the received message, and sends the character count back to the client. It also handles potential errors that may occur during socket creation, binding, listening, accepting connections, reading data, and writing data.

CLIENT CODE client.c

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

int main()
{
   int cli_sock = socket(AF_INET, SOCK_STREAM, 0);
   if (cli_sock == -1)
   {
      perror("Error creating client socket");
      exit(1);
   }

   // Initialize server address structure
   struct sockaddr_in servaddr;
   bzero(&servaddr, sizeof(servaddr));
   servaddr.sin_family = AF_INET;
   servaddr.sin_port = htons(8080);
   servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");

   // Connect to the server
   if (connect(cli_sock, (struct sockaddr *)&servaddr, sizeof(servaddr)) == -1)
   {
      perror("Error connecting to the server");
      close(cli_sock);
      exit(1);
   }

   // Open the file for reading
   FILE *fp = fopen("file.txt", "r");
   if (fp == NULL)
   {
      perror("Error opening file for reading");
      close(cli_sock);
      exit(1);
   }

   // Calculate file size
   fseek(fp, 0, SEEK_END);
   int fileSize = ftell(fp);
   fseek(fp, 0, SEEK_SET);
```

```
    // Allocate memory for file contents
    char *fileContents = (char *)malloc(fileSize + 1);
    if (fileContents == NULL)
    {
        perror("Error allocating memory for file contents");
        fclose(fp);
        close(cli_sock);
        exit(1);
    }

    // Read file contents into the buffer
    size_t end = fread(fileContents, 1, fileSize, fp);
    fileContents[end] = '\0';
    fclose(fp);

    // Send file contents to the server
    if (send(cli_sock, fileContents, fileSize, 0) == -1)
    {
        perror("Error sending file contents to the server");
        free(fileContents);
        close(cli_sock);
        exit(1);
    }

    // Receive the count of characters from the server
    int receivedCount;
    if (read(cli_sock, &receivedCount, sizeof(receivedCount)) == -1)
    {
        perror("Error receiving character count from the server");
        free(fileContents);
        close(cli_sock);
        exit(1);
    }

    // Convert receivedCount from network byte order to host byte order
    receivedCount = ntohl(receivedCount);

    printf("Received count of characters from server: %d\n", receivedCount - 1); // To avoid the null
character at the end of the string to be counted.

    // Clean up and close the client socket
    free(fileContents);
    close(cli_sock);
    return 0;
}
```

1. **Socket Creation:**
   - It creates a client socket using the `socket()` function with the address family `AF_INET` (IPv4), socket type `SOCK_STREAM` (for TCP), and protocol `0` (default protocol).

2. **Server Address Initialization:**
   - It initializes a `struct sockaddr_in` named `servaddr` to represent the server's address.
   - It sets the address family to `AF_INET`.
   - It specifies the server's port number using `htons()` to convert it to network byte order.
   - It sets the server's IP address to "127.0.0.1," which is the loopback address (localhost).

3. **Connecting to the Server:**
   - It connects to the server using the `connect()` function, specifying the client socket, server address structure, and its size.
   - If the connection fails, it handles the error and exits.

4. **File Reading:**
   - It opens a file named "file.txt" for reading using the `fopen()` function.
   - If the file opening fails, it handles the error and exits.

5. **File Size Calculation:**
   - It calculates the size of the file using `fseek()` and `ftell()` to determine the number of bytes in the file.

6. **Memory Allocation:**
   - It allocates memory for storing the file contents in the `fileContents` buffer using `malloc()`.
   - If memory allocation fails, it handles the error and exits.

7. **File Contents Reading:**
   - It reads the contents of the file into the `fileContents` buffer using the `fread()` function.
   - It ensures that the contents are null-terminated by setting `fileContents[end]` to '\0'.
   - It then closes the file using `fclose()`.

8. **Data Transmission:**
   - It sends the file contents to the server using the `send()` function.
   - If sending data to the server fails, it handles the error and exits.

9. **Receiving Data from Server:**
   - It receives the character count from the server using the `read()` function.
   - It also performs network byte order to host byte order conversion using `ntohl()` to get the correct value.
   - The character count represents the number of characters in the received data.

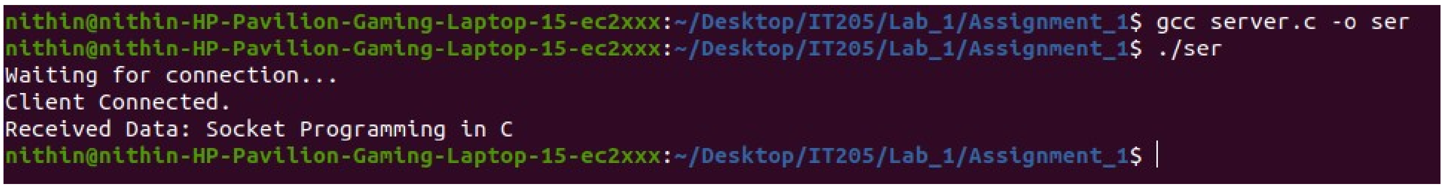
10. **Displaying the Character Count:**
    - It displays the received character count on the client's console.
    - To avoid counting the null character at the end of the string, it subtracts 1 from the count.

11. **Cleaning Up:**
    - It releases the allocated memory for `fileContents` using `free()`.
    - It closes the client socket using `close()`.

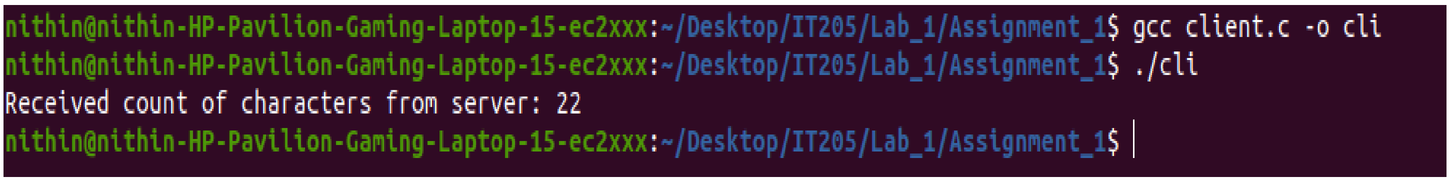
The client code establishes a connection to the server, reads the contents of "file.txt," sends them to the server, receives a character count from the server, and then displays the character count on the client's console. It also handles potential errors that may occur during socket creation, connection, file reading, data sending, and data receiving.

File to read : **file.txt** has the text **“Socket Programming in C”** which has 22 characters(excluding the \0 character)..This will be read by Client and be sent to the Server by Client and Server will return the no. Of characters in it to the Client.

<u>OUTPUT</u>



```
nithin@nithin-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~/Desktop/IT205/Lab_1/Assignment_1$ gcc server.c -o ser
nithin@nithin-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~/Desktop/IT205/Lab_1/Assignment_1$ ./ser
Waiting for connection...
Client Connected.
Received Data: Socket Programming in C
nithin@nithin-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~/Desktop/IT205/Lab_1/Assignment_1$
```



```
nithin@nithin-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~/Desktop/IT205/Lab_1/Assignment_1$ gcc client.c -o cli
nithin@nithin-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~/Desktop/IT205/Lab_1/Assignment_1$ ./cli
Received count of characters from server: 22
nithin@nithin-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~/Desktop/IT205/Lab_1/Assignment_1$
```

Here the Client is reading the contents of the text file file.txt and sending it to Server and Server sends back the No. Of characters in that content sent.