Peter D

```
void send_file(int clientSocket, struct sockaddr_in &serverAddr) {
    ifstream inFile("input.bmp", ios::binary);
    char buffer[SIZE];
    socklen_t addrlen = sizeof(serverAddr);

    if (!inFile) {
        cerr << "Error opening file for reading." << endl;
        return;
    }

    cout << "Sending file..." << endl;
    while (inFile.read(buffer, SIZE) || inFile.gcount() > 0) {
        sendto(clientSocket, buffer, inFile.gcount(), 0, (struct sockaddr *)&serverAddr, addrlen);
    }
    inFile.close();
    cout << "File sent successfully." << endl;
}
```

Opening(binary)and sending 1024-byte chunks of data (one packet) to the server while the condition is met.

```
int main() {
    int clientSocket;
    struct sockaddr_in serverAddr, clientAddr;
    char message[] = "HELLO";
    char buffer[SIZE];
    socklen_t addrlen = sizeof(serverAddr);

    clientSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (clientSocket < 0) {
        perror("Socket creation failed");
        return 1;
    }

    memset(&clientAddr, 0, sizeof(clientAddr));
    clientAddr.sin_family = AF_INET;
    clientAddr.sin_addr.s_addr = INADDR_ANY;
    clientAddr.sin_port = htons(CLIENT_PORT);

    if (bind(clientSocket, (const struct sockaddr *)&clientAddr, sizeof(clientAddr)) < 0) {
        perror("Bind failed");
        close(clientSocket);
        return 1;
    }

    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(SERVER_PORT);

    sendto(clientSocket, message, strlen(message), 0, (struct sockaddr *)&serverAddr, addrlen);
    cout << "Sent: " << message << endl;

    ssize_t recvLen = recvfrom(clientSocket, buffer, SIZE, 0, (struct sockaddr *)&serverAddr, &addrlen);
    if (recvLen < 0) {
        perror("Receive failed");
    } else {
        buffer[recvLen] = '\0';
        cout << "Received echo: " << buffer << endl;
    }

    char startMessage[] = "START_FILE_TRANSFER";
    sendto(clientSocket, startMessage, strlen(startMessage), 0, (struct sockaddr *)&serverAddr, addrlen);
    send_file(clientSocket, serverAddr);

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

Implementation of the UDP client using the Beej's Guide to Network Programming and the Linux Tutorial on Socket Programming as references

```cpp
void receive_file(int serverSocket, struct sockaddr_in &clientAddr, socklen_t &addrlen) {
    ofstream outFile("received.bmp", ios::binary);
    char buffer[SIZE];
    ssize_t recvLen;

    if (!outFile) {
        cerr << "Error opening file for writing." << endl;
        return;
    }

    cout << "Receiving file..." << endl;
    while ((recvLen = recvfrom(serverSocket, buffer, SIZE, 0, (struct sockaddr *)&clientAddr, &addrlen)) > 0) {
        outFile.write(buffer, recvLen);
        if (recvLen < SIZE) break;
    }
    outFile.close();
    cout << "File received successfully." << endl;
}
```

Server receiving the packets from the client one packet at a time until its completed.

```cpp
int main() {
    int serverSocket;
    char buffer[SIZE];
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t addrlen = sizeof(clientAddr);

    serverSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (serverSocket < 0) {
        perror("Socket creation failed");
        return 1;
    }

    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(SERVER_PORT);

    if (bind(serverSocket, (const struct sockaddr *)&serverAddr, sizeof(serverAddr)) < 0) {
        perror("Bind failed");
        close(serverSocket);
        return 1;
    }

    cout << "UDP Server listening on port " << SERVER_PORT << endl;

    while (true) {
        ssize_t recvLen = recvfrom(serverSocket, buffer, SIZE, 0,
                                   (struct sockaddr *)&clientAddr, &addrlen);
        if (recvLen < 0) {
            perror("Receive failed");
            break;
        }
        buffer[recvLen] = '\0';
        cout << "Received from client (port " << ntohs(clientAddr.sin_port) << "): " << buffer << endl;

        if (strcmp(buffer, "START_FILE_TRANSFER") == 0) {
            receive_file(serverSocket, clientAddr, addrlen);
            continue;
        }

        sendto(serverSocket, buffer, recvLen, 0, (struct sockaddr *)&clientAddr, addrlen);
        cout << "Echoed back to client (port " << ntohs(clientAddr.sin_port) << "): " << buffer << endl;
    }

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

Implementation of the UDP server using the Beej's Guide to Network Programming and the Linux Tutorial on Socket Programming as references