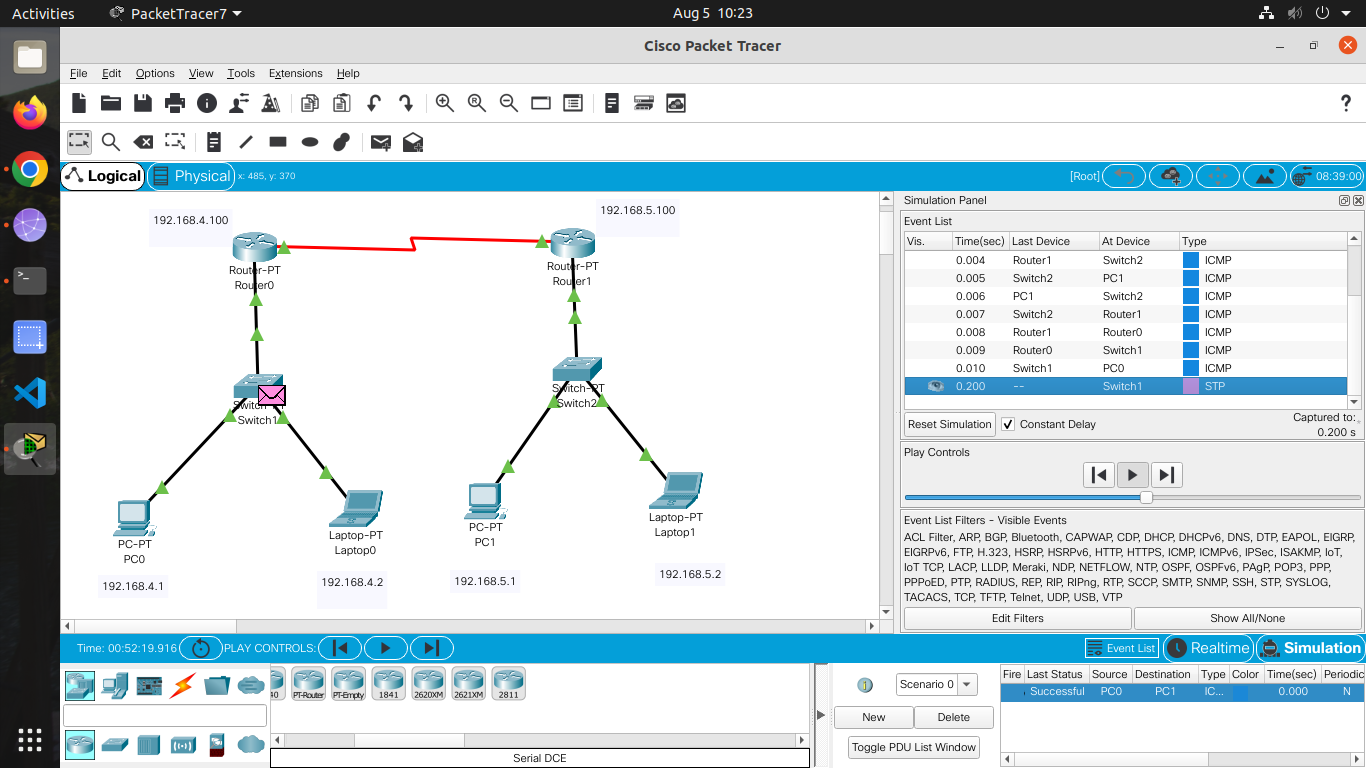
**Name:** *Tejas Padhiyar*

**Roll No.:** *31451*

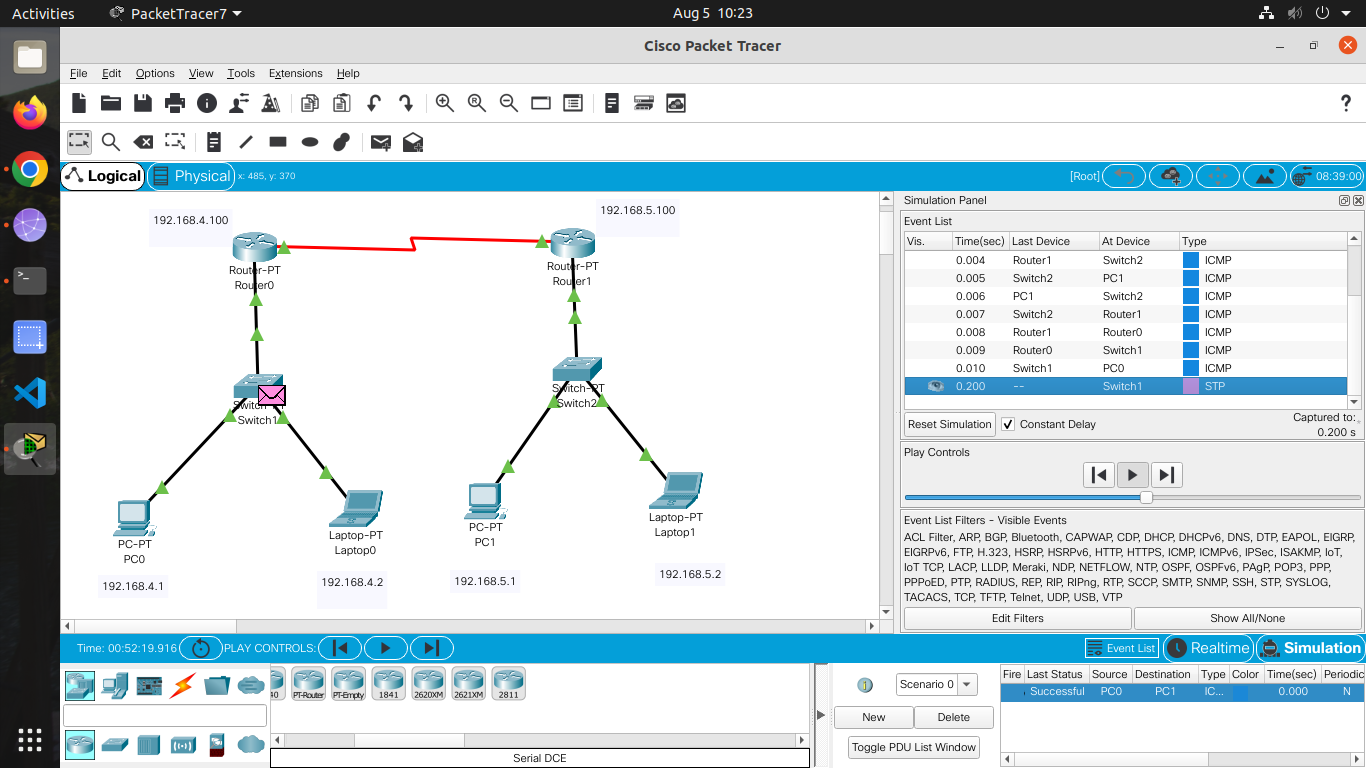
**Batch:** *M4*

**Class:** *TE4*

**Assignment No. 1 (A2)**



**Assignment No. 2 (A2)**



**Assignment No. 3 (A3)**

#include <bits/stdc++.h>

using namespace std;

char generate\_parity\_1(string bin, int n)

{

    int counter = 0;

    for (int i = 0; i < n; i++)

    {

        if (bin[i] == '1')

        {

            counter++;

        }

    }

    if (counter % 2 == 0)

        return '0';

    else

        return '1';

}

char generate\_parity(char bin[], int n)

{

    int counter = 0;

    for (int i = 0; i < n; i++)

    {

        if (bin[i] == '1')

            counter++;

    }

    if (counter % 2 == 0)

        return '0';

    else

        return '1';

}

void reverse(char str[], int length)

{

    int start = 0;

    int end = length -1;

    while (start < end)

    {

        swap(\*(str+start), \*(str+end));

        start++;

        end--;

    }

}

char\* itoa(int num, char\* str, int base)

{

    int i = 0;

    bool isNegative = false;

    /\* Handle 0 explicitly, otherwise empty string is printed for 0 \*/

    if (num == 0)

    {

        str[i++] = '0';

        str[i] = '\0';

        return str;

    }

    // In standard itoa(), negative numbers are handled only with

    // base 10. Otherwise numbers are considered unsigned.

    if (num < 0 && base == 10)

    {

        isNegative = true;

        num = -num;

    }

    // Process individual digits

    while (num != 0)

    {

        int rem = num % base;

        str[i++] = (rem > 9)? (rem-10) + 'a' : rem + '0';

        num = num/base;

    }

    // If number is negative, append '-'

    if (isNegative)

        str[i++] = '-';

    str[i] = '\0'; // Append string terminator

    // Reverse the string

    reverse(str, i);

    return str;

}

int main()

{

    string s;

    cout << "Enter a String :" << endl;

    cin >> s;

    char p1, p2, p4, p8, hamming[11], p1\_a[5], p2\_a[5], p4\_a[3], p8\_a[3];

    for (int i = 0; i < s.length(); i++)

    {

        int c\_ascii = int(s[i]);

        char bin[7];

        char bin\_cpy[7];

        cout << "Character:" << s[i] << endl;

        cout << "Decimal:" << c\_ascii << endl;

        cout << "Binary:";

        itoa(c\_ascii, bin, 2);

        // bin\_cpy[0]='0';

        int counter = 0;

        for (int i = 0; i <= 7; i++)

        {

            bin\_cpy[i] = bin[counter];

            counter++;

        }

        for (int i = 0; i < 7; i++)

        {

            cout << bin\_cpy[i];

        }

        cout << endl;

        for (int i = 0; i < 5; i++)

        {

            if (i == 0)

            {

                p1\_a[i] = bin\_cpy[0];

                p2\_a[i] = bin\_cpy[0];

                p4\_a[i] = bin\_cpy[1];

                p8\_a[i] = bin\_cpy[4];

            }

            else if (i == 1)

            {

                p1\_a[i] = bin\_cpy[1];

                p2\_a[i] = bin\_cpy[2];

                p4\_a[i] = bin\_cpy[2];

                p8\_a[i] = bin\_cpy[5];

            }

            else if (i == 2)

            {

                p1\_a[i] = bin\_cpy[3];

                p2\_a[i] = bin\_cpy[3];

                p4\_a[i] = bin\_cpy[3];

                p8\_a[i] = bin\_cpy[6];

            }

            else if (i == 3)

            {

                p1\_a[i] = bin\_cpy[4];

                p2\_a[i] = bin\_cpy[5];

            }

            else

            {

                p1\_a[i] = bin\_cpy[6];

                p2\_a[i] = bin\_cpy[6];

            }

        }

        p1 = generate\_parity(p1\_a, 5);

        p2 = generate\_parity(p2\_a, 5);

        p4 = generate\_parity(p4\_a, 3);

        p8 = generate\_parity(p8\_a, 3);

        hamming[0] = p1;

        hamming[1] = p2;

        int counter1 = 0;

        for (int i = 2; i < 11; i++)

        {

            if (i == 3)

                hamming[i] = p4;

            else if (i == 7)

                hamming[i] = p8;

            else

            {

                hamming[i] = bin\_cpy[counter1];

                counter1++;

            }

        }

        cout << endl;

        cout << "p1:" << p1 << endl;

        cout << "p2:" << p2 << endl;

        cout << "p4:" << p4 << endl;

        cout << "p8:" << p8 << endl;

        cout << "Do you want to corrupt the data word ? (y/n)" << endl;

        char ch;

        bool flag;

        cin >> ch;

        if (ch == 'y')

        {

            flag = true;

            cout << "Enter the bit position to corrupt:" << endl;

            int pos;

            cin >> pos;

            if (hamming[pos - 1] == '1')

            {

                hamming[pos - 1] = '0';

            }

            else

            {

                hamming[pos - 1] = '1';

            }

            cout << "Corrupted Code Word: ";

            for (int i = 0; i < 11; i++)

            {

                cout << hamming[i];

            }

            cout << endl;

        }

        else

        {

            flag = false;

            cout << "Uncorrupted Code Word:";

            for (int i = 0; i < 11; i++)

            {

                cout << hamming[i];

            }

        }

        cout << endl;

        cout << "RECEIVER SIDE" << endl;

        string p1\_check = "";

        string p2\_check = "";

        string p4\_check = "";

        string p8\_check = "";

        p1\_check = p1\_check + hamming[0] + hamming[2] + hamming[4] + hamming[6] + hamming[8] + hamming[10]; // 0 2 4 6 8 10

        p2\_check = p2\_check + hamming[1] + hamming[2] + hamming[5] + hamming[6] + hamming[9] + hamming[10]; // 1 2 5 6 9 10

        p4\_check = p4\_check + hamming[3] + hamming[4] + hamming[5] + hamming[6]; // 3 4 5 6

        p8\_check = p8\_check + hamming[7] + hamming[8] + hamming[9] + hamming[10]; // 7 8 9 10

        char p1\_rec = generate\_parity\_1(p1\_check, 6);

        char p2\_rec = generate\_parity\_1(p2\_check, 6);

        char p4\_rec = generate\_parity\_1(p4\_check, 4);

        char p8\_rec = generate\_parity\_1(p8\_check, 4);

        cout << "p1:" << p1\_rec << endl;

        cout << "p2:" << p2\_rec << endl;

        cout << "p4:" << p4\_rec << endl;

        cout << "p8:" << p8\_rec << endl;

        string pos = "";

        pos = pos + p8\_rec + p4\_rec + p2\_rec + p1\_rec;

        int pos\_no = stoi(pos, nullptr, 2);

        cout << pos\_no << endl;

        if(flag == true)

        {

            if (hamming[pos\_no - 1] == '1')

            {

                hamming[pos\_no - 1] = '0';

            }

            else

            {

                hamming[pos\_no - 1] = '1';

            }

            cout<<"Corrected Code Word is : ";

            for(int k = 0; k <= 11; k++)

            {

                cout<<hamming[k];

            }

        }

        cout<<endl<<"Data Word : ";

        for(int k = 0; k <= i; k++)

        {

            cout<<s[k];

        }

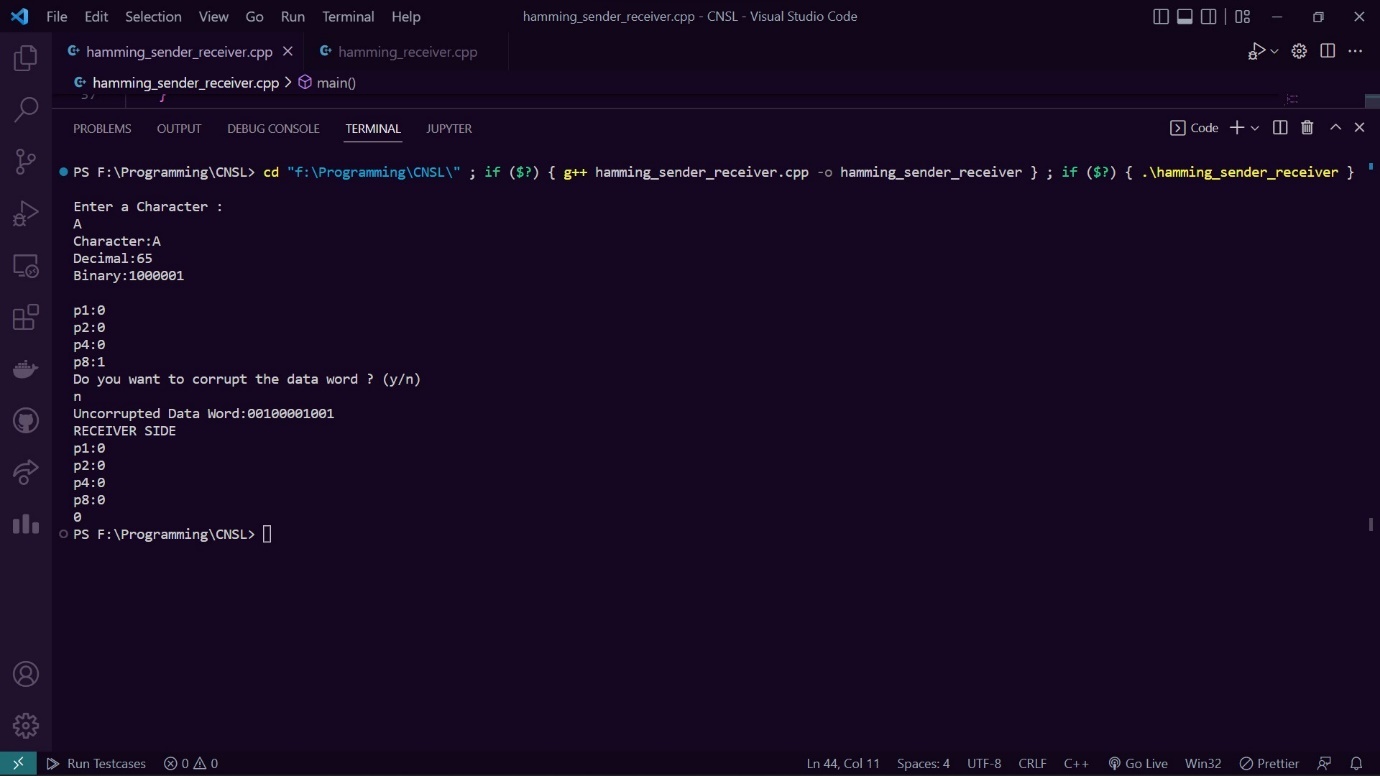
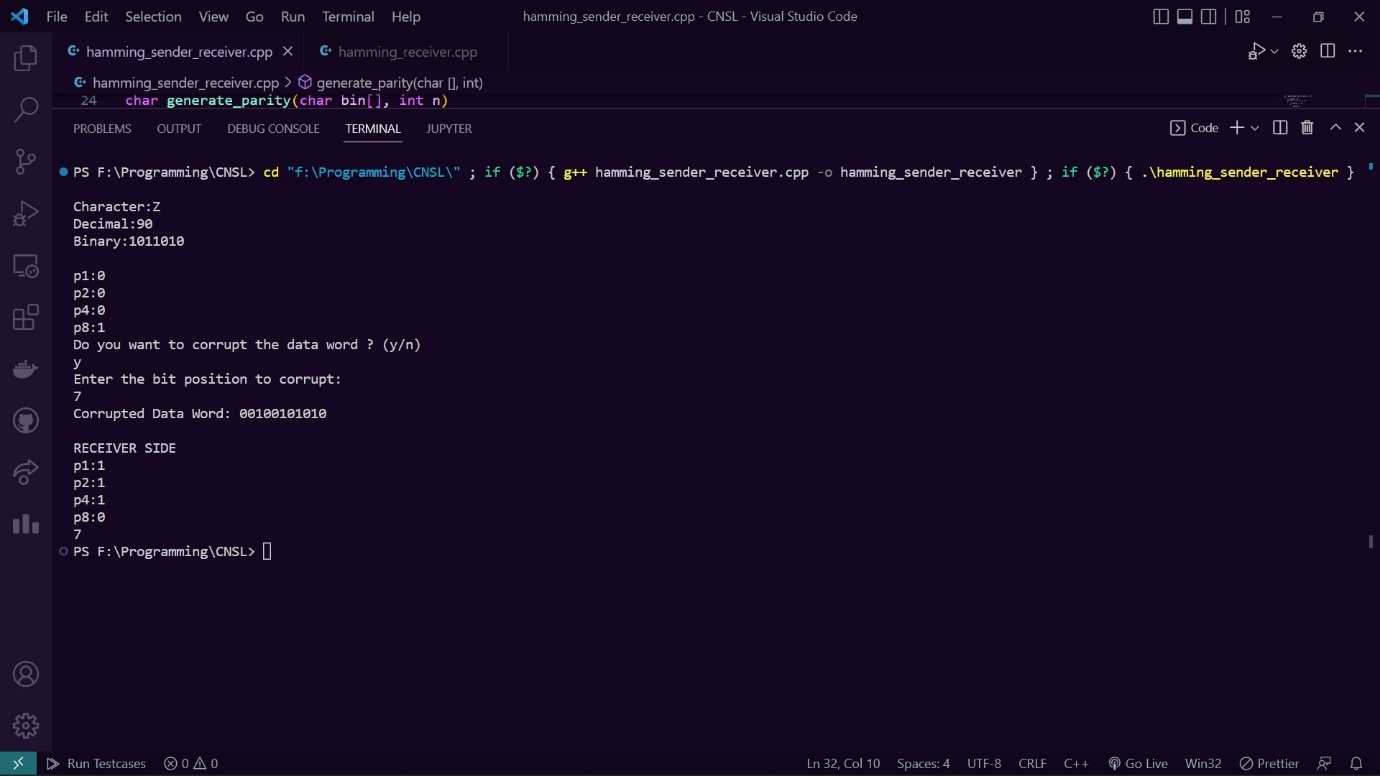
        cout<<endl<<"========================="<<endl;

    }

    return 0;

}

**Output:**



**Assignment No. 4 (A4)**

Server.cpp

#include <bits/stdc++.h>

#include <sys/socket.h>

#include <cstring>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <thread>

#include <chrono>

#include <iostream>

using namespace std;

void gbn(int);

void sr(int);

int m;

int min\_seq\_num = 0;

int max\_seq\_num;

int current\_sequence\_number = min\_seq\_num;

int acknowledgement\_remaining = min\_seq\_num;

int size\_of\_sliding\_window;

int maximum\_sequence\_number;

struct msg {

        char data;

        int sequence\_number;

};

struct rmsg {

    bool isAck;

    int sequence\_number;

};

int main() {

    cout<<"--------------SERVER SIDE--------------"<<endl;

    int sfd,cfd;

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

    if (sfd == -1) {

        cout << "socket not created" << endl;;

        exit(1);

    }

    struct sockaddr\_in my\_addr,peer\_addr;

    memset(&my\_addr, 0, sizeof(struct sockaddr\_in));

    my\_addr.sin\_family = AF\_INET;

    my\_addr.sin\_port = htons(8080);

    inet\_aton("0.0.0.0",&my\_addr.sin\_addr);

    if (bind(sfd, (struct sockaddr \*) &my\_addr, sizeof(struct sockaddr\_in)) == -1) {

        cout << "error in binding" << endl;

        exit(1);

    }

    if (listen(sfd,50) == -1) {

        cout << "error in listening" << endl;

        exit(1);

    }

    socklen\_t peer\_addr\_size;

    peer\_addr\_size = sizeof(struct sockaddr\_in);

    int choice = 0;

    cout << "Sliding Window Protocols: \n1.GO Back N\n2.Selective Repeat"<<endl;

        cout << "Enter your choice : ";

        cin >> choice;

        if (choice == 1) {

        cout << "Enter the size of bit sequence (m) : " ;

            cin >> m;

        max\_seq\_num = pow(2,m) - 1;

        cout << "Sequence number possible from " << min\_seq\_num << " to " << max\_seq\_num << endl;

        while (true) {

            cfd = accept(sfd, (struct sockaddr \*) &peer\_addr, &peer\_addr\_size);

            cout << cfd <<  " connected ip " << inet\_ntoa(peer\_addr.sin\_addr) << ":" << peer\_addr.sin\_port << endl;

                    gbn(cfd);

        }

        }

        else if (choice == 2) {

        cout << "Enter the size of bit sequence (m) : ";

        cin >> m;

        max\_seq\_num = pow(2,m) - 1;

        size\_of\_sliding\_window = pow(2, m - 1);

            maximum\_sequence\_number = current\_sequence\_number + size\_of\_sliding\_window - 1;

        cout << "Sequence number possible from " << min\_seq\_num << " to " << max\_seq\_num << endl;

        cout << "Receiver sliding window size : " << size\_of\_sliding\_window << endl;

        cout << "From " << min\_seq\_num << " to " << maximum\_sequence\_number << endl;

        while (true) {

            cfd = accept(sfd, (struct sockaddr \*) &peer\_addr, &peer\_addr\_size);

            cout << cfd <<  " connected ip " << inet\_ntoa(peer\_addr.sin\_addr) << ":" << peer\_addr.sin\_port << endl;

                    sr(cfd);

        }

        }

    return 0;

}

void gbn(int cfd) {

    int expected\_sequence\_number = min\_seq\_num;

        std::random\_device dev;

        std::mt19937 rng(dev());

        std::uniform\_int\_distribution<std::mt19937::result\_type> distBin(0,1);

    while (true) {

        msg m1;

        int received\_sequence\_number;

        int res = recv(cfd, &m1, sizeof(m1), 0);

        if (res == 0) {

            break;

            return;

        }

        received\_sequence\_number = m1.sequence\_number;

        cout << "data received : " << m1.data << endl;

        cout << "received frame number " << received\_sequence\_number << " while expecting " << expected\_sequence\_number << endl;

        if (distBin(rng)) {

            cout << "Randomly discarding this frame" << endl;

            continue;

        }

        if (received\_sequence\_number == expected\_sequence\_number) {

            cout << "frame received correctly" << endl;

            expected\_sequence\_number = (expected\_sequence\_number + 1) % (max\_seq\_num + 1);

        };

        cout << "requesting for frame number " << expected\_sequence\_number << endl;

        cout<<endl;

        // this\_thread::sleep\_for(chrono::seconds(2));

        send(cfd, &expected\_sequence\_number, sizeof(int), 0);

    }

}

void sr(int cfd) {

    vector<pair<int,bool>> backlog;

    bool nakSent = false;

    for (int i = min\_seq\_num; i <= maximum\_sequence\_number; i++) {

        backlog.push\_back({i,false});

    }

    std::random\_device dev;

        std::mt19937 rng(dev());

        std::uniform\_int\_distribution<std::mt19937::result\_type> distBin(0,1);

    while (true) {

        msg m1;

        int received\_sequence\_number;

        for (int i = 0; i < backlog.size(); i++) {

            cout << backlog[i].first << " ";

        }

        cout << endl;

        int res = recv(cfd, &m1, sizeof(m1), 0);

        if (res == 0) {

            break;

            return;

        }

        received\_sequence\_number = m1.sequence\_number;

        cout << "data received : " << m1.data << endl;

        cout << "received frame number " << received\_sequence\_number << " while expecting " << backlog.begin()->first << endl;

        if (distBin(rng)) {

            cout << "Randomly discarding this frame" << endl;

            continue;

        }

        for (int i = 0; i < backlog.size(); i++) {

            if (backlog[i].first == received\_sequence\_number) {

                backlog[i].second = true;

                break;

            }

        }

        if (!nakSent && !backlog.begin()->second) {

            cout << "sending Negative Acknowledgement" << endl;

            int nak = backlog.begin()->first;

            rmsg m1;

            m1.sequence\_number = nak;

            m1.isAck = false;

            send(cfd, &m1, sizeof(m1), 0);

            nakSent = true;

        }

        int i = 0;

        while (backlog[i].second) {

            i = (i + 1) % (max\_seq\_num+1);

        }

        if (backlog[((i-1) + (max\_seq\_num + 1)) % (max\_seq\_num+1)].first == received\_sequence\_number) {

            cout << "sending Acknowledgement" << endl;

            int ack = (backlog[((i-1) + (max\_seq\_num + 1)) % (max\_seq\_num+1)].first + 1) % (max\_seq\_num + 1);

            rmsg m1;

            m1.sequence\_number = ack;

            m1.isAck = true;

            send(cfd, &m1, sizeof(m1), 0);

            for (int i = 0; i < backlog.size(); i++) {

                if (backlog[i].second) {

                    backlog.erase(backlog.begin() + i);

                    backlog.push\_back({(backlog.back().first+1) % (max\_seq\_num + 1), false});

                    i--;

                }

            }

            cout<<endl;

        }

    }

}

Client.cpp

#include <bits/stdc++.h>

#include <sys/socket.h>

#include <cstring>

#include <netinet/in.h>

#include <netinet/ip.h> /\* superset of previous \*/

#include <arpa/inet.h>

#include <chrono>

#include <thread>

#include <iostream>

using namespace std;

void gbn(int);

void sr(int);

struct msg {

    char data;

    int sequence\_number;

};

struct rmsg {

    bool isAck;

    int sequence\_number;

};

int main() {

    cout<<"--------------CLIENT SIDE--------------"<<endl;

    int cfd;

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

    if (cfd == -1) {

        cout << "socket not created" << endl;;

        exit(1);

    }

    struct sockaddr\_in my\_addr,peer\_addr;

    memset(&my\_addr, 0, sizeof(struct sockaddr\_in));

    my\_addr.sin\_family = AF\_INET;

    my\_addr.sin\_port = htons(0);

    inet\_aton("127.0.0.1",&my\_addr.sin\_addr);

    memset(&peer\_addr, 0, sizeof(struct sockaddr\_in));

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

    peer\_addr.sin\_port = htons(8080);

    inet\_aton("127.0.0.1", &peer\_addr.sin\_addr);

    // my\_addr.sin\_addr=(in\_addr)INADDR\_LOOPBACK;

    if (bind(cfd, (struct sockaddr \*) &my\_addr, sizeof(struct sockaddr\_in)) == -1) {

        cout << "error in binding" << endl;

        exit(errno);

    }

    if (connect(cfd, (struct sockaddr \*) &peer\_addr, sizeof(peer\_addr)) == -1) {

        cout << "error in connecting" << endl;

        exit(errno);

    }

    cout << "Connected to the server! (127.0.0.1)" << endl;

    int choice = 0;

    cout << "Sliding Window Protocols: \n1.GO Back N\n2.Selective Repeat"<<endl;

    cout << "Enter your choice : ";

    cin >> choice;

    if (choice == 1) {

        gbn(cfd);

    }

    else if (choice == 2) {

        sr(cfd);

    }

    return 0;

}

int m;

int min\_seq\_num = 0;

int max\_seq\_num;

int current\_sequence\_number = min\_seq\_num;

int acknowledgement\_remaining = min\_seq\_num;

int size\_of\_sliding\_window;

int maximum\_sequence\_number;

int acknowledgedDataIndex = -1;

mutex m1;

void sendFramesGBN(int cfd, string data) {

    unique\_lock<mutex> l(m1,defer\_lock);

    int dataIndex = 0;

    bool flagDataSent = false;

    while (true) {

        for (int i = 0; i < size\_of\_sliding\_window - 1 ; i++) {

            if (dataIndex >= data.size()) {

                flagDataSent = true;

                break;

            }

            cout << "Sending frame " << data[dataIndex] << " with sequence number " << current\_sequence\_number << endl;

            l.lock();

            msg m1 = msg {data[dataIndex++], current\_sequence\_number};

            send(cfd, &m1, sizeof(m1), 0);

            current\_sequence\_number = (current\_sequence\_number + 1) % (size\_of\_sliding\_window + 1);

            l.unlock();

            cout << "Frame sent" << endl;

            this\_thread::sleep\_for(chrono::seconds(1));

        }

        l.lock();

        if (acknowledgement\_remaining != current\_sequence\_number) {

            cout << "Waiting for acknowledgement for frame number " << acknowledgement\_remaining << endl;

            cout << "waiting for 3 seconds" << endl;

            this\_thread::sleep\_for(chrono::seconds(3));

            if (acknowledgement\_remaining != current\_sequence\_number) {

                cout << "resending frames, starting from frame number " << acknowledgement\_remaining << endl;

                dataIndex = acknowledgedDataIndex+1;

                flagDataSent = false;

                current\_sequence\_number = acknowledgement\_remaining;

            }

        } else if (flagDataSent) {

            // close(cfd);

            exit(0);

        }

        l.unlock();

    }

    cout<<endl;

}

bool check(int a, int b, int c) {

    if (a < b) {

        if (a < c && c < b) return true;

        else return false;

    } else {

         if (b < c && c < a) return false;

      else return true;

    }

}

void recvAcksGBN(int cfd) {

    unique\_lock<mutex> l(m1,defer\_lock);

    struct timeval tv;

    fd\_set cfds;

    FD\_ZERO(&cfds);

    FD\_SET(cfd, &cfds);

    tv.tv\_sec = 1;

    while (true) {

        int ack;

        recv(cfd, &ack, sizeof(int), 0);

        cout << "acknowledgment received, requesting number " << ack << endl;

        //cout << "->" << min\_seq\_num << " " << maximum\_sequence\_number << endl;

        if (check(min\_seq\_num, maximum\_sequence\_number, ack)) {

            l.lock();

            acknowledgedDataIndex++;

            int number\_of\_frames\_acknowledged = abs(ack - acknowledgement\_remaining) % (size\_of\_sliding\_window-1);

            acknowledgement\_remaining = ack;

            //cout << "acknowledgement\_remaining changed to " << acknowledgement\_remaining << endl;

            min\_seq\_num = ack;

            maximum\_sequence\_number = (maximum\_sequence\_number + number\_of\_frames\_acknowledged) % (size\_of\_sliding\_window + 1);

            l.unlock();

        }

    }

    cout<<endl;

}

void gbn(int cfd) {

    cout << "Enter the size of bit sequence (m): " ;

    cin >> m;

    max\_seq\_num = pow(2,m) - 1;

    size\_of\_sliding\_window = pow(2, m) - 1;

    maximum\_sequence\_number = current\_sequence\_number + size\_of\_sliding\_window - 1;

    cout << "Sequence number possible from " << min\_seq\_num << " to " << max\_seq\_num << endl;

    cout << "Size of the sliding window is " << size\_of\_sliding\_window << endl;

    cout << "Current Sliding window " << acknowledgement\_remaining << " to " << maximum\_sequence\_number << endl;

    cout << "Enter the data to be sent : ";

    string data;

    cin >> data;

    std::random\_device dev;

    std::mt19937 rng(dev());

    std::uniform\_int\_distribution<std::mt19937::result\_type> distBin(0,1);

    // std::cout << distBin(rng) << std::endl;

    thread t2(sendFramesGBN, cfd, data);

    thread t1(recvAcksGBN, cfd);

    t2.join();

    t1.join();

}

vector<pair<int,bool>> receivedAcknowledgments;

vector<char> chars;

int dataIndex = 0;

int dataIndexTemp = 0;

void sendFramesSR(int cfd, string data) {

    unique\_lock<mutex> l(m1,defer\_lock);

    bool flagDataSent = false;

    while (true) {

        for (int i = 0; i < size\_of\_sliding\_window ; i++) {

            if (!receivedAcknowledgments[i].second) {

                cout << "Sending frame " << chars[current\_sequence\_number] << " with sequence number " << current\_sequence\_number << endl;

                l.lock();

                msg m1 = msg {chars[current\_sequence\_number], current\_sequence\_number};

                send(cfd, &m1, sizeof(m1), 0);

                current\_sequence\_number = (current\_sequence\_number + 1) % (max\_seq\_num + 1);

                l.unlock();

                cout << "Frame sent" << endl;

                this\_thread::sleep\_for(chrono::seconds(1));

                if (dataIndexTemp == data.size()+size\_of\_sliding\_window) {

                    flagDataSent = true;

                    break;

                }

            }

        }

        if (flagDataSent) {

            exit(0);

        }

        bool allNotReceived = true;

        for (int i = 0; i < receivedAcknowledgments.size(); i++) {

            if (receivedAcknowledgments[i].second) {

                allNotReceived = false;

                break;

            }

        }

        l.lock();

        if (!allNotReceived) {

            for (int i = 0; i < receivedAcknowledgments.size(); i++) {

                if (!receivedAcknowledgments[i].second) {

                    cout << "Waiting for acknowledgement for frame having sequence number " << receivedAcknowledgments[i].first << endl;

                    cout << "waiting for 3 seconds" << endl;

                    this\_thread::sleep\_for(chrono::seconds(3));

                    if (!receivedAcknowledgments[i].second) {

                        cout << "resending frames, starting from frame number " << receivedAcknowledgments[i].first << endl;

                        dataIndex = acknowledgedDataIndex+1;

                        flagDataSent = false;

                        current\_sequence\_number = receivedAcknowledgments[i].first;

                        break;

                    }

                }

            }

        }

        cout<<endl;

        l.unlock();

    }

}

void recvAcksSR(int cfd, string data) {

    unique\_lock<mutex> l(m1,defer\_lock);

    struct timeval tv;

    fd\_set cfds;

    FD\_ZERO(&cfds);

    FD\_SET(cfd, &cfds);

    tv.tv\_sec = 1;

    while (true) {

        rmsg ack;

        recv(cfd, &ack, sizeof(ack), 0);

        if (ack.isAck) {

            cout << "Acknowledgement Received " << ack.sequence\_number << endl;

            l.lock();

            for (int i = 0; i < receivedAcknowledgments.size(); i++) {

                if (receivedAcknowledgments[i].first == ack.sequence\_number) {

                    break;

                }

                receivedAcknowledgments.erase(receivedAcknowledgments.begin() + i);

                maximum\_sequence\_number = (maximum\_sequence\_number + 1) % (max\_seq\_num + 1);

                if (dataIndex+1 <= data.size()) {

                    chars[maximum\_sequence\_number] = data[dataIndex++];

                }

                dataIndexTemp++;

                if (dataIndex == data.size()+size\_of\_sliding\_window) {

                    dataIndex = dataIndex-1;

                }

                receivedAcknowledgments.push\_back({maximum\_sequence\_number, false});

                acknowledgedDataIndex++;

                i--;

            }

            cout << "sliding window shifted : " << endl;

            for (int i = 0; i < receivedAcknowledgments.size(); i++) {

                cout << receivedAcknowledgments[i].first << " ";

            }

            cout << endl;

            l.unlock();

        } else {

            cout << "Negaive Acknowledgement Received" << endl;

            l.lock();

            for (int i = 0; i < receivedAcknowledgments.size(); i++) {

                if (ack.sequence\_number == receivedAcknowledgments[i].first) {

                    receivedAcknowledgments[i].second = false;

                }

            }

            l.unlock();

        }

    }

    cout<<endl;

}

void sr(int cfd) {

    cout << "Enter the size of bit sequence (m): ";

    cin >> m;

    max\_seq\_num = pow(2, m) - 1;

    size\_of\_sliding\_window = pow(2, m - 1);

    maximum\_sequence\_number = current\_sequence\_number + size\_of\_sliding\_window - 1;

    cout << "Sequence number possible from " << min\_seq\_num << " to " << max\_seq\_num << endl;

    cout << "Size of the sliding window is " << size\_of\_sliding\_window << endl;

    cout << "Current Sliding window " << acknowledgement\_remaining << " to " << maximum\_sequence\_number << endl;

    cout << "Enter the data you want to send : ";

    string data;

    vector<char> temp(max\_seq\_num + 1);

    chars = temp;

    cin >> data;

    dataIndex = 0;

    for (int i = min\_seq\_num; i <= maximum\_sequence\_number; i++) {

        chars[i] = data[dataIndex++];

        dataIndexTemp++;

        receivedAcknowledgments.push\_back({i,false});

    }

    thread t2(sendFramesSR, cfd, data);

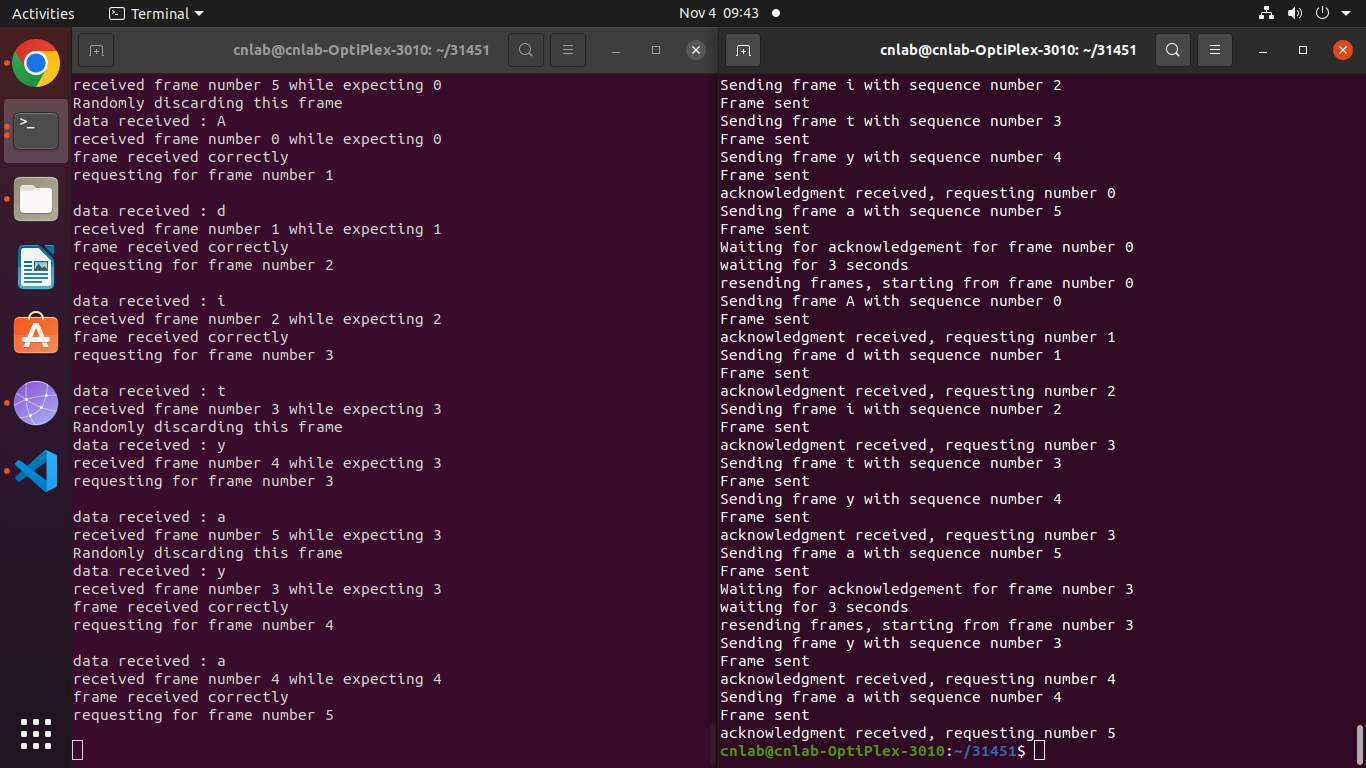
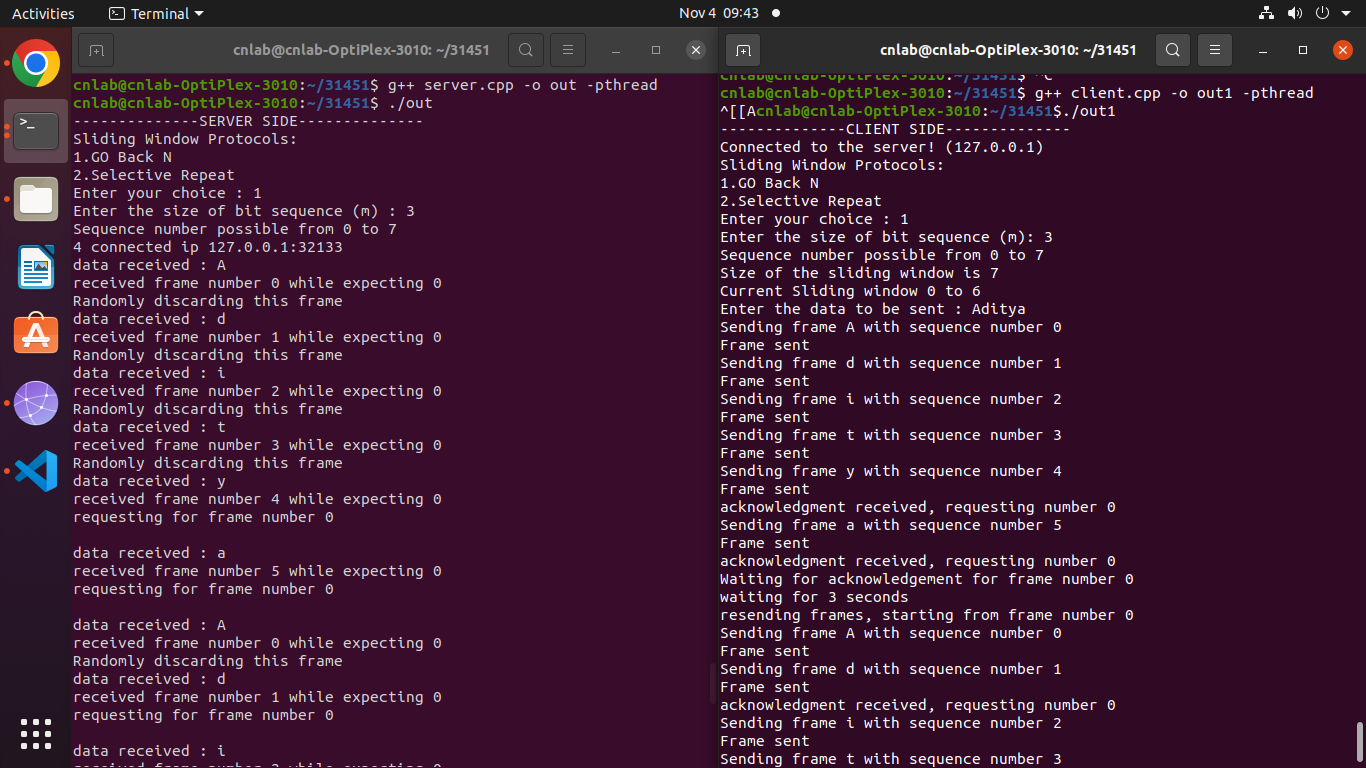
    thread t1(recvAcksSR, cfd, data);

    t2.join();

    t1.join();

}

**Output:**



**Assignment No. 5 (B1)**

import java.util.\*;

import java.lang.Math;

public class Subnetting

{

    // 8+8+8+x

    // powerNumber = 2^x

    int powerNumber;

    private int getPowerNumber()

    {

        return powerNumber;

    }

    private void setPowerNumberFromNoOfSubnets( int nSubnets )

    {

        while( 256%nSubnets != 0 )

        {

            nSubnets++;

        }

        // powerNumber = (int)Math.pow(2,nSubnets);

        powerNumber = 256/nSubnets;

    }

    private void setPowerNumberFromCIDR( int cidr )

    {

        // finding 8 + 8 + ? + ?

        int mod = cidr%8;

        powerNumber = (int)Math.pow(2,8-mod);

    }

    private int getNumberOfSubnets()

    {

        return (256/powerNumber);

    }

    public static void main(String[] args)

    {

        Scanner sc = new Scanner(System.in);

        String ip,subnetMask = "255.255.";

        int choice,cidr,nSubnets;

        boolean isSupernetting = false;

        Subnetting subnetting = new Subnetting(); // created object because main() is static. Either do this or create another class especially for main()

        System.out.println("Enter Ip Address");

        ip = sc.next();

        String[] test = ip.split("\\.",5);

        for ( String str : test )

        {

            int x = Integer.valueOf(str);

            if( x < 0 || x > 255 )

            {

                System.out.println("Invalid IP");

                System.exit(1);

            }

        }

        System.out.println("1. Enter CIDR ( ex. 26 )");

        System.out.println("2. Enter number of subnets ( ex. 4 )");

        choice = sc.nextInt();

        if( choice!=1 && choice !=2)

        {

            System.out.println("Invalid Input");

            sc.close();

            System.exit(1);

        }

        if( choice == 1 )

        {

            cidr = sc.nextInt();

            if( cidr < 16 || cidr > 31)

            {

                System.out.println("CIDR Does not fit into subnetting or supernetting");

                System.exit(1);

            }

            // finding if supernetting or subnetting

            if( Integer.valueOf(cidr / 8) < 3 )

                isSupernetting = true;

            subnetting.setPowerNumberFromCIDR(cidr);

        }

        else if ( choice == 2 )

        {

            nSubnets = sc.nextInt();

            subnetting.setPowerNumberFromNoOfSubnets(nSubnets);

        }

        int host = 256 - subnetting.getPowerNumber();

        if( isSupernetting )

            subnetMask += host + ".0";

        else

            subnetMask += "255." + host;

        System.out.println(subnetMask);

        if(!isSupernetting)

            System.out.println("Number of subnets formed: " + subnetting.getNumberOfSubnets());

        else

            System.out.println("Number of supernets formed: " + subnetting.getNumberOfSubnets());

        // removing last element from

        ArrayList<String> test2 = new ArrayList<>(Arrays.asList(test));

        int lastIpBits;

        if( isSupernetting )

        {

            test2.remove(2);

            test2.remove(2);

            lastIpBits = Integer.valueOf(test[2]);

        }

        else

        {

            test2.remove(3);

            lastIpBits = Integer.valueOf(test[3]);

        }

        // converting array back to string

        // half ip will be first 3 ip bits e.g. 192.168.13. ( for printing range )

        String halfIp = "";

        for( String str : test2 )

        {

            halfIp = halfIp + str + ".";

        }

        // finding range

        int pow = subnetting.getPowerNumber();

        int maxLimit = pow;

        int minLimit = 0;

        while( 256 >= maxLimit )

        {

            if( !isSupernetting )

                System.out.print( halfIp + minLimit + " to " + halfIp + (maxLimit-1) );

            else

                System.out.print( halfIp + minLimit + ".0" + " to " + halfIp + (maxLimit-1) + ".0");

            if( minLimit < lastIpBits && maxLimit > lastIpBits )

                System.out.print(" <- ip belongs to this range\n");

            else

                System.out.println();

            minLimit = maxLimit;

            maxLimit += pow;

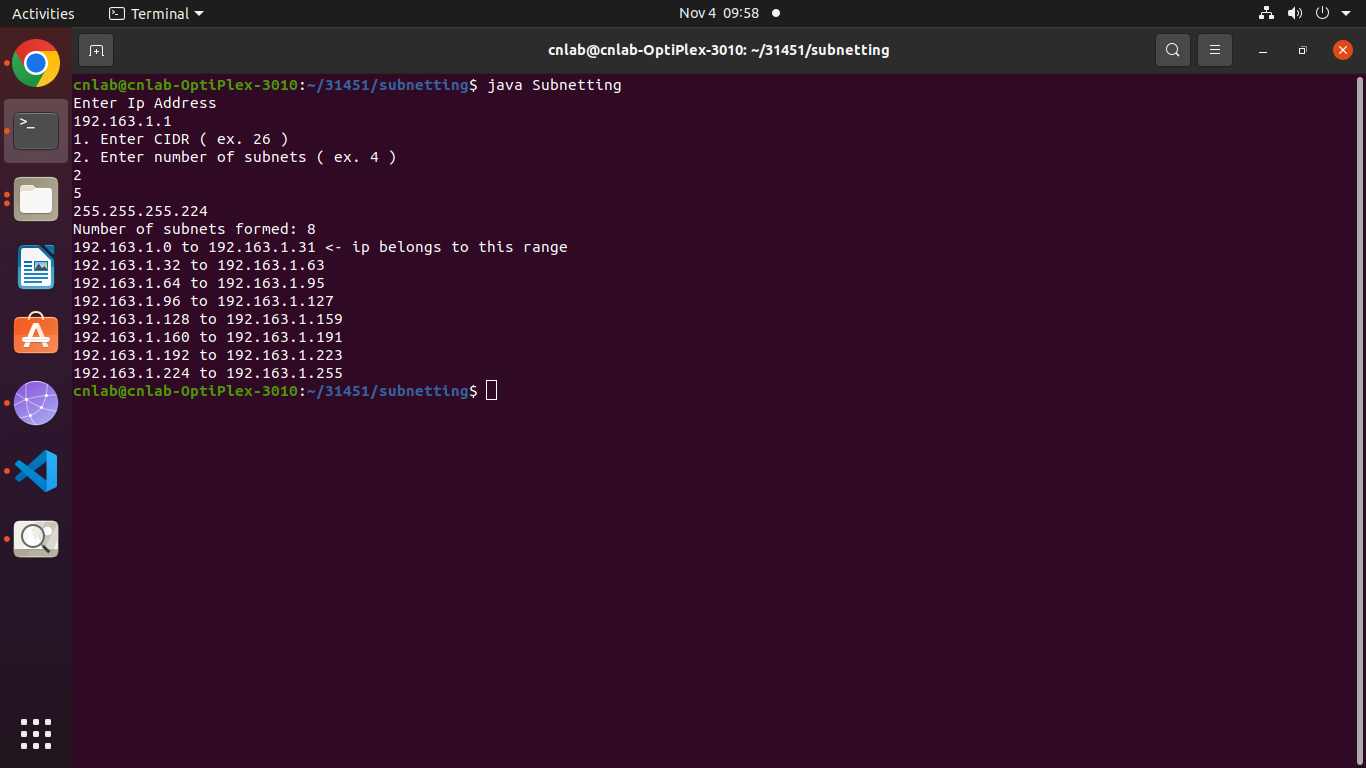
        }

        sc.close();

    }

}

**Output:**



**Assignment No. 6 (B2)**

#include<stdio.h>

#include<iostream>

using namespace std;

struct node

{

    unsigned dist[6];

    unsigned from[6];

}DVR[10];

int main()

{

    cout<<"\n\n-------------------- Distance Vector Routing Algorithm----------- ";

    int costmat[6][6];

    int nodes, i, j, k;

    cout<<"\n\n Enter the number of nodes : ";

    cin>>nodes; //Enter the nodes

    cout<<"\n Enter the cost matrix : \n" ;

    for(i = 0; i < nodes; i++)

     {

        for(j = 0; j < nodes; j++)

        { cout<<"Enter value at "<<i <<" --"<<j<<" : ";

            cin>>costmat[i][j];

            costmat[i][i] = 0;

            DVR[i].dist[j] = costmat[i][j]; //initialise the distance equal to cost matrix

            DVR[i].from[j] = j;

        }

    }

            for(i = 0; i < nodes; i++) //We choose arbitary vertex k and we calculate the

            //direct distance from the node i to k using the cost matrix and add the distance from k to node j

            for(j = i+1; j < nodes; j++)

            for(k = 0; k < nodes; k++)

                if(DVR[i].dist[j] > costmat[i][k] + DVR[k].dist[j])

                { //We calculate the minimum distance

                    DVR[i].dist[j] = DVR[i].dist[k] + DVR[k].dist[j];

                    DVR[j].dist[i] = DVR[i].dist[j];

                    DVR[i].from[j] = k;

                    DVR[j].from[i] = k;

                }

        for(i = 0; i < nodes; i++)

        {

            cout<<"\n\n For router: "<<i+1;

            for(j = 0; j < nodes; j++)

                cout<<"\t\n node "<<j+1<<" via "<<DVR[i].from[j]+1<<" Distance "<<DVR[i].dist[j];

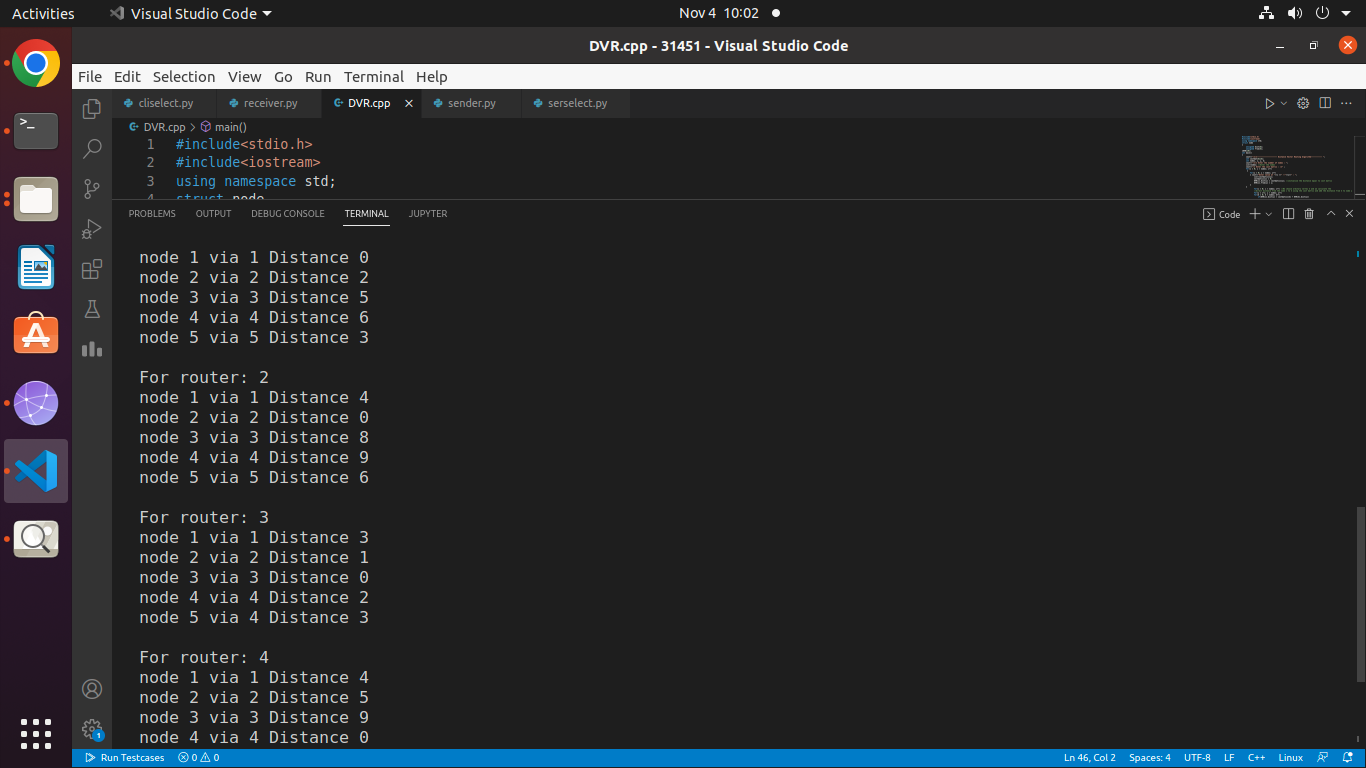
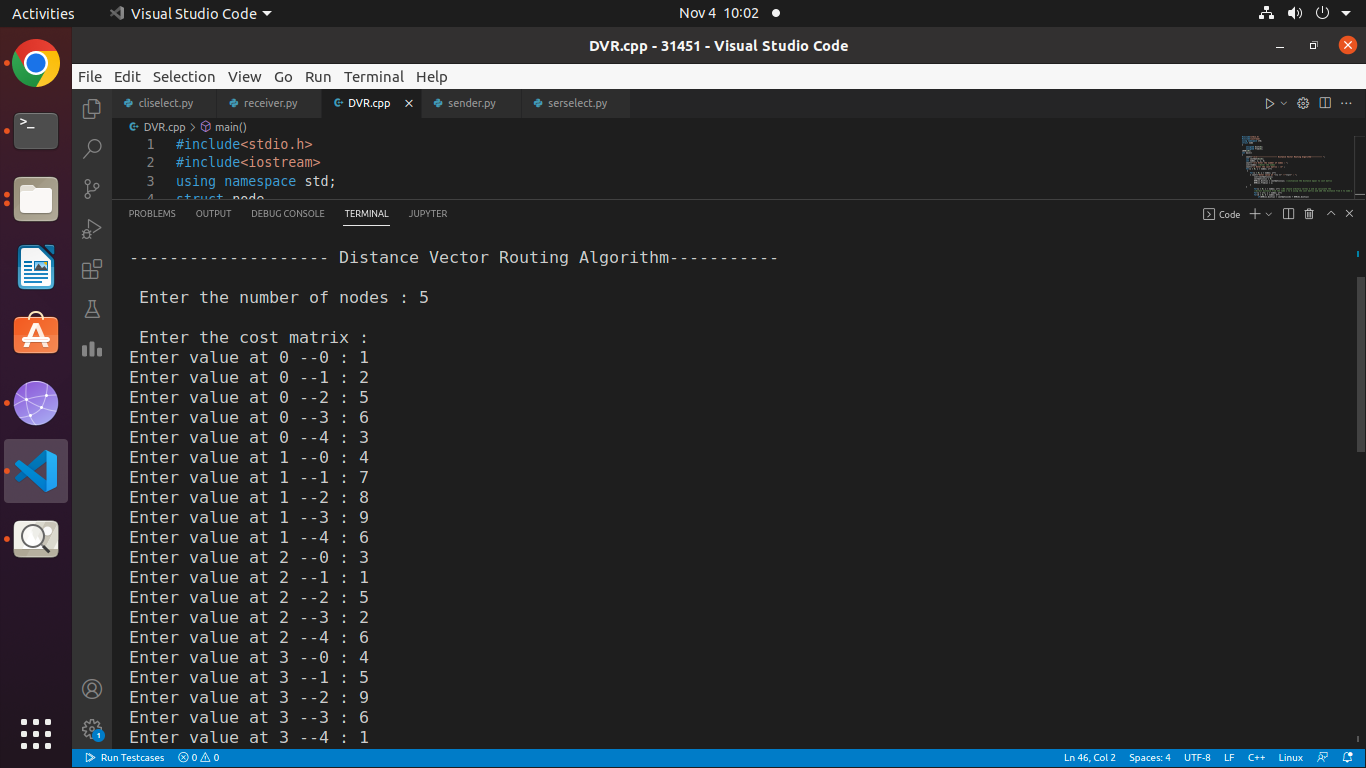
        }

    cout<<" \n\n ";

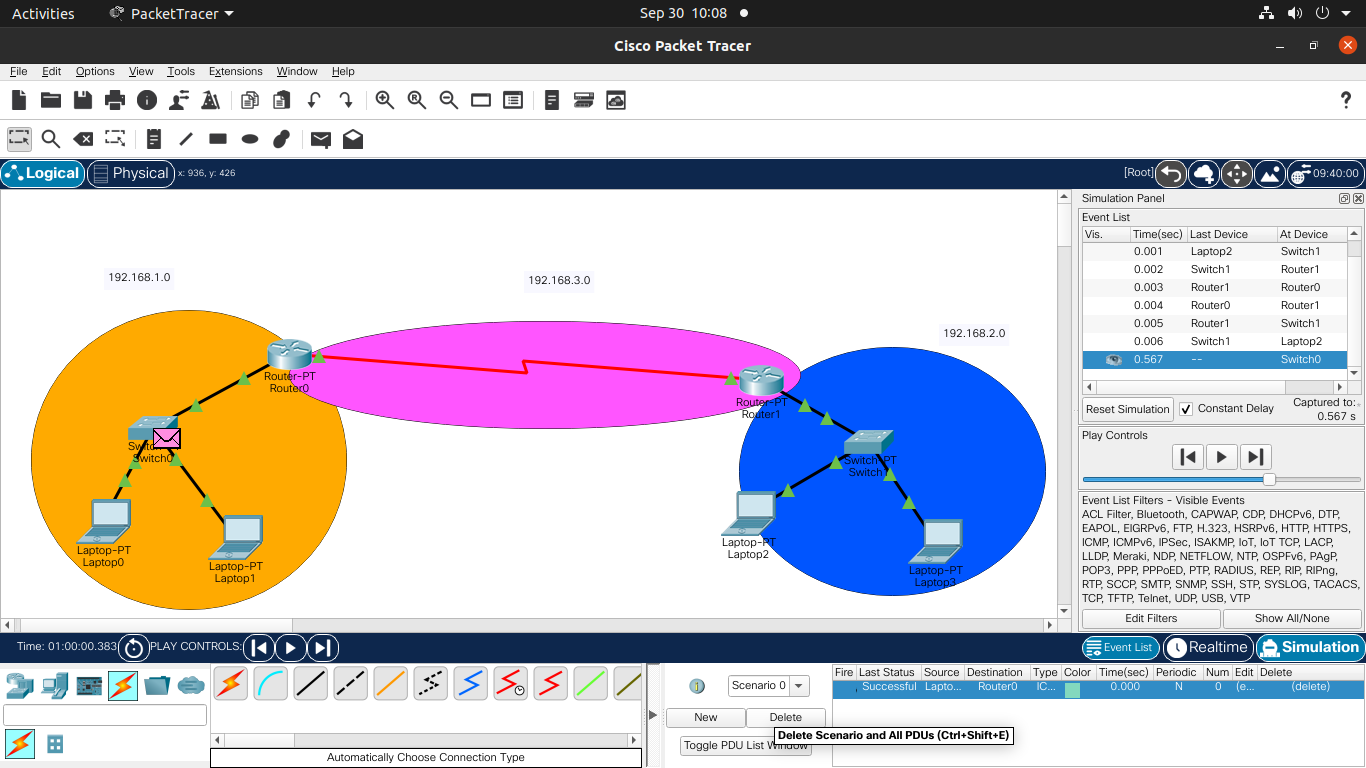
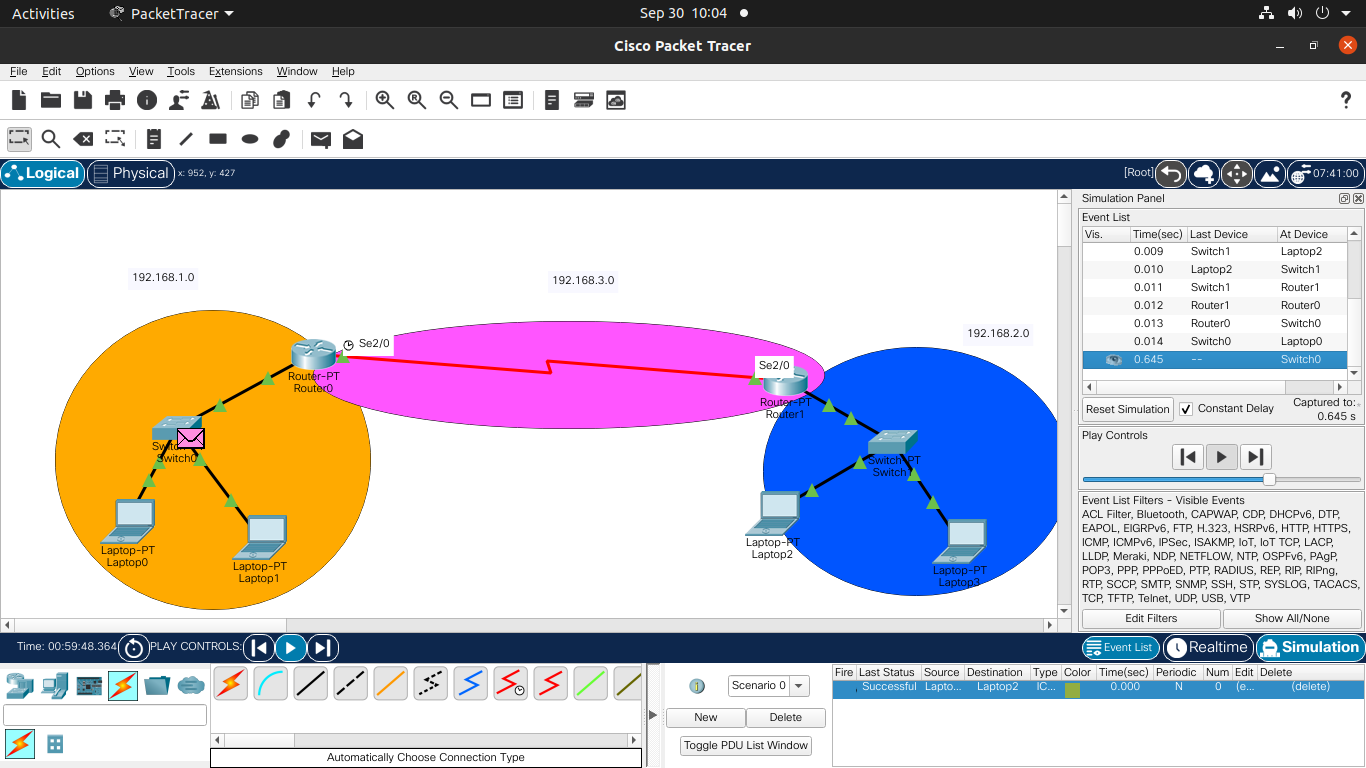
    return 0;

}

**Output:**



**Assignment No. 7 (B3)**



**Assignment No. 8 (B4)**

Server.cpp

#include<iostream>

#include<unistd.h>

#include<netinet/in.h>

#include<string.h>

#include<string>

#include<fstream>

#include <arpa/inet.h>

#include<sys/socket.h>

#define PORT 6511

#define MAXLINES 1024

using namespace std;

int main()

{

    int sockfd,connfd;

    sockaddr\_in server,client;

    char buffer[MAXLINES];

    char fileBuffer[MAXLINES];

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

    server.sin\_family = AF\_INET;

    server.sin\_port = htons(PORT);

    server.sin\_addr.s\_addr = INADDR\_ANY;

    bind(sockfd,(const sockaddr\*) &server,sizeof(server));

    listen(sockfd,5);

    socklen\_t len;

    connfd = accept(sockfd, (sockaddr \*)&client, &len);

     if (connfd < 0)

         cout << "failse";

     else

         cout << "success";

    int n = read(connfd,buffer,sizeof(buffer));

    buffer[n] = '\0';

    cout << "Client said: " << buffer << endl;

    ifstream ifs(buffer,ios::in|ios::ate);

    int size = ifs.tellg();

    ifs.seekg(ios::beg);

    ifs.read(fileBuffer,size);

    send(connfd,fileBuffer,size,0);

    return 0;

}

Client.cpp

#include<iostream>

#include<unistd.h>

#include<netinet/in.h>

#include<string.h>

#include<string>

#include<fstream>

#include <arpa/inet.h>

#include<sys/socket.h>

#define PORT 6511

#define MAXLINES 1024

using namespace std;

int main()

{

    int sockfd;

    sockaddr\_in server;

    char buffer[MAXLINES];

    string fileName;

    if( (sockfd = socket(AF\_INET,SOCK\_STREAM,0) ) < 0 )

    {

        cout << "err";

    }

    server.sin\_family = AF\_INET;

    server.sin\_port = htons(PORT);

    server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    if(connect(sockfd,(const sockaddr \*) &server,sizeof(server)) < 0 )

        cout << "error";

    cout << "Enter fileName" << endl;

    cin >> fileName;

    send(sockfd,fileName.c\_str(),fileName.length(),0);

    int n = read(sockfd,buffer,MAXLINES);

    ofstream ofs(fileName,ios::out);

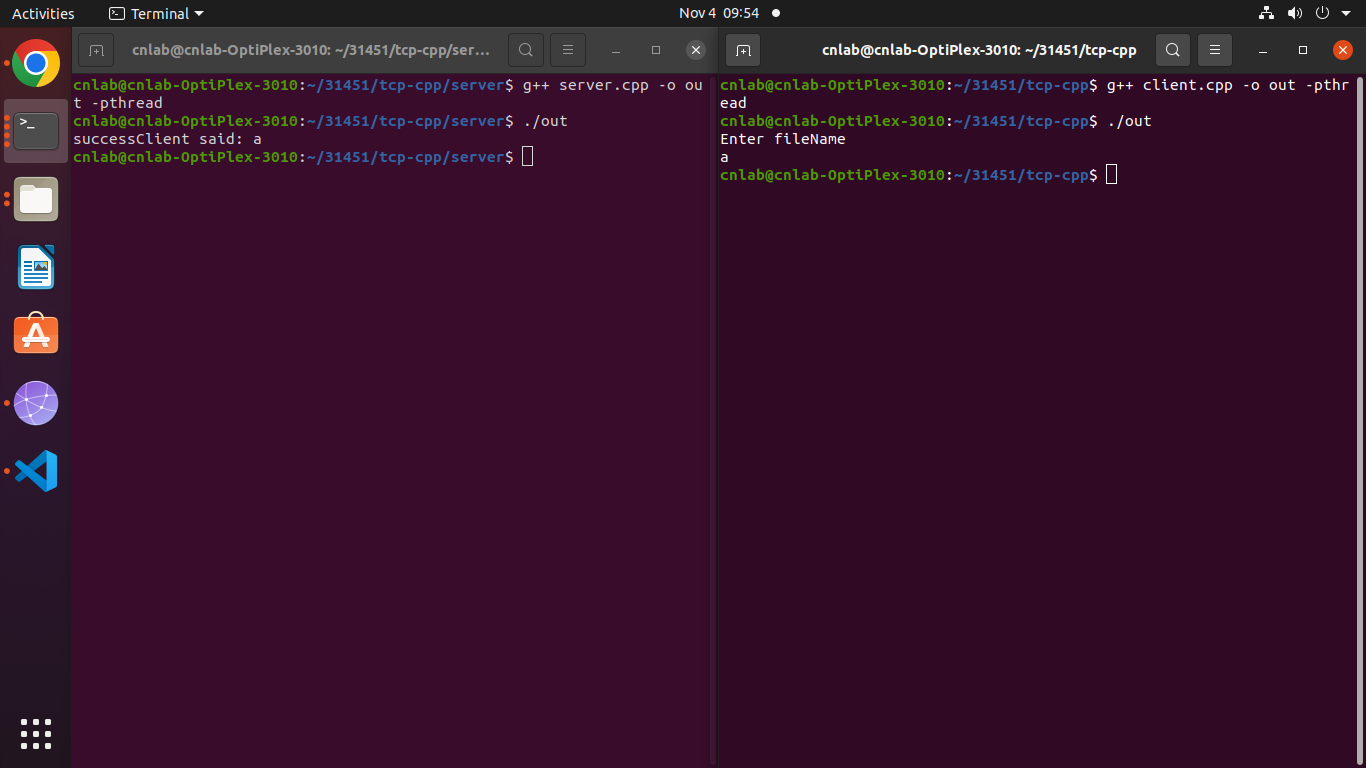
    ofs.write(buffer,n);

    ofs.close();

    return 0;

}

**Output:**



**Assignment No. 9 (B5)**

Server.cpp

#include<iostream>

#include<stdlib.h>

#include<netinet/in.h>

#include<unistd.h>

#include<string.h>

#include<sys/socket.h>

#include<stdio.h>

#include<fstream>

#include<string>

#define PORT 7512

#define MAXLINES 1024

using namespace std;

int main()

{

    int sockfd;

    char\* fileName = new char[1];

    char buffer[MAXLINES];

    struct sockaddr\_in server,client;

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

    memset(&server,0,sizeof(server));

    memset(&client,0,sizeof(client));

    server.sin\_family = AF\_INET;

    server.sin\_addr.s\_addr = INADDR\_ANY;

    server.sin\_port = htons(PORT);

    bind(sockfd,(const struct sockaddr\*) &server,sizeof(server));

    socklen\_t len;

    int n = recvfrom(sockfd,fileName,MAXLINES,0,( struct sockaddr \*)&client,&len);

    fileName[n]='\0';

    cout << "Client wants file: " << fileName << endl;

    cout << "Opening file" << endl;

    std::ifstream ifs(fileName,ios::ate);

    if(!ifs)

    {

        cout << "file not present";

    }

    else

    {

        int size = ifs.tellg();

        ifs.seekg(ios::beg);

        cout << "Reading file. size:"<<size << endl;

        ifs.read(buffer,size);

        cout << "Sending file" << endl;

        sendto(sockfd,(const char \*)buffer,size,0,(const struct sockaddr \*) &client,sizeof(client));

        cout << "file sent " << endl;

        cout << "Closing file" << endl;

        ifs.close();

        cout << "Closing socket" << endl;

        close(sockfd);

    }

    return 0;

}

Client.cpp

#include<iostream>

#include<fstream>

#include<stdlib.h>

#include<netinet/in.h>

#include<unistd.h>

#include<string.h>

#include<sys/socket.h>

#include<stdio.h>

#include<string>

#define PORT 7512

#define MAXLINES 1024

using namespace std;

int main()

{

    int sockfd;

    string fileName;

    char buffer[MAXLINES];

    struct sockaddr\_in server;

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

    memset(&server,0,sizeof(server));

    server.sin\_family = AF\_INET;

    server.sin\_addr.s\_addr = INADDR\_ANY;

    server.sin\_port = htons(PORT);

    cout << "Enter file name" << endl;

    cin >> fileName;

    cout << fileName.c\_str() << endl;

    sendto(sockfd, (const char \*)fileName.c\_str(), fileName.length(),0,(const struct sockaddr \*)&server, sizeof(server));

    cout << "Sent filename to server" << endl;

    cout << "Waiting for response" << endl;

    socklen\_t len;

    int n = recvfrom(sockfd,(char \*)buffer,sizeof(buffer),0,(struct sockaddr \*)&server,&len);

    buffer[n] = '\0';

    cout << "Response recieved: " << buffer << endl;

    ofstream ofs;

    cout << "Creating file" << endl;

    ofs.open(fileName,ios::out);

    cout << "Writing to file" << endl;

    ofs.write(buffer,n);

    cout << "Closing file" << endl;

    ofs.close();

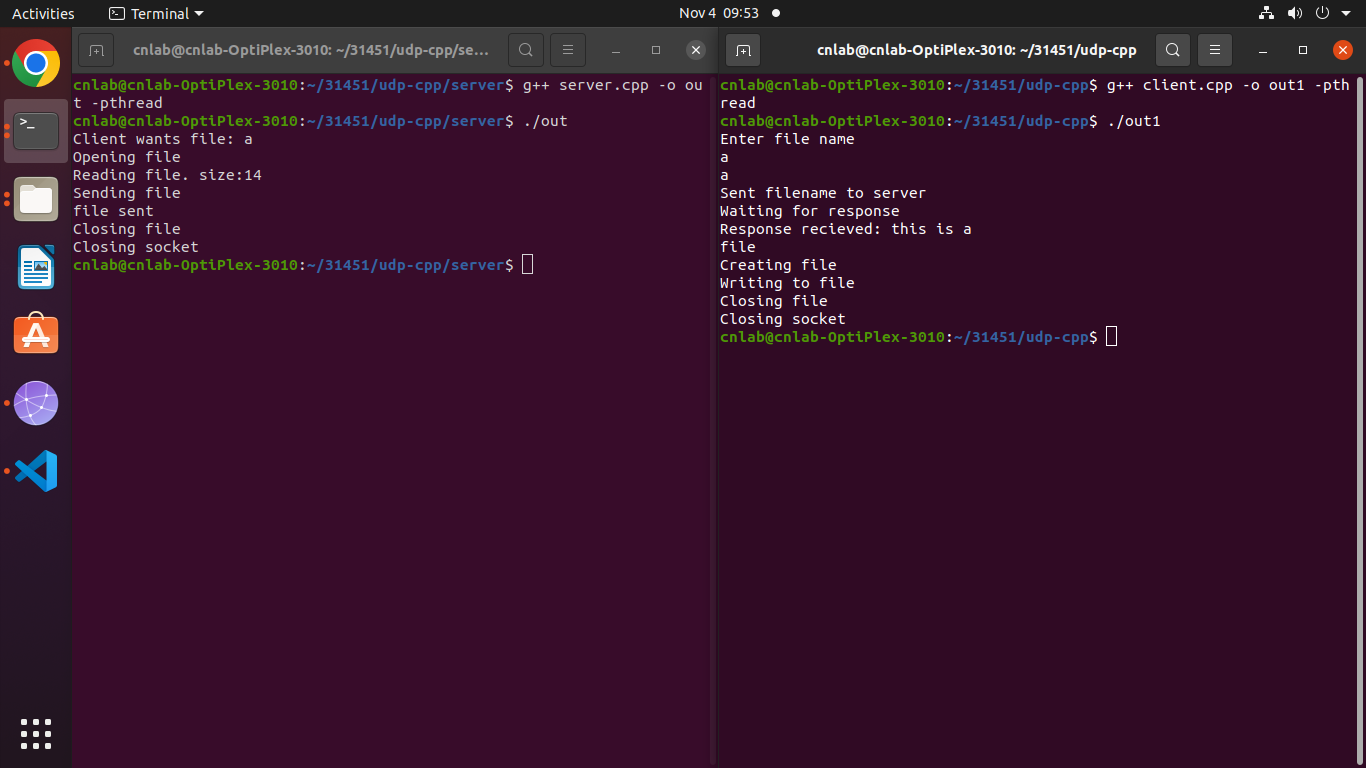
    cout << "Closing socket"<<endl;

    close(sockfd);

    return 0;

}

**Output:**



**Assignment No. 10 (C1)**

import java.net.\*;

import java.util.\*;

public class dns

{

 public static void main(String[] args){

  String host;

  Scanner ch = new Scanner(System.in);

  System.out.print("1.Enter Host Name \n2.Enter IP address \nChoice=");

  int choice = ch.nextInt();

  if(choice==1)

  {

  Scanner input = new Scanner(System.in);

  System.out.print("\n Enter host name: ");

  host = input.nextLine();

  try {

   InetAddress address = InetAddress.getByName(host);

   System.out.println("IP address: " + address.getHostAddress());

   System.out.println("Host name : " + address.getHostName());

   System.out.println("Host name and IP address: " + address.toString());

  }

  catch (UnknownHostException ex) {

       System.out.println("Could not find " + host);

  }

  }

  else

  {

  Scanner input = new Scanner(System.in);

  System.out.print("\n Enter IP address: ");

  host = input.nextLine();

  try {

   InetAddress address = InetAddress.getByName(host);

   System.out.println("Host name : " + address.getHostName());

   System.out.println("IP address: " + address.getHostAddress());

   System.out.println("Host name and IP address: " + address.toString());

  }

  catch (UnknownHostException ex) {

       System.out.println("Could not find " + host);

  }

  }

 }

}

**Output:**

