Experiment 1: Error Detection using Parity

Aim: To apply Parity check rules for error detection

Objective: After carrying out this experiment, students will be able to:

- Apply 1D and 2D parity rules for error detection
- Analyze the difference between 1D and 2D parity and their limitations

Problem statement: You are required to write separate programs to demonstrate the use of 1D and 2D parity. Take the input bit streams (max five) of 7 bit each from the user. Your programs should calculate the parity and display the input and output bit streams.

Analysis: While analyzing your program, you are required to address the following points:

- Why can this method not be used to correct errors?
- How are 1D and 2D parity different?
- What are the limitations of this method of error detection?

MARKS DISTRIBUTION

Component	Maximum Marks	Marks Obtained
Preparation of Document	7	
Results	7	
Viva	6	
Total	**20**	

Submitted by: Satyajit Ghana

Register No: 17ETCS002159

1. Algorithm/Flowchart

```
1D-Parity(data):
1. bit_count = 0
2. for_each(bit : data)
3.    if (bit == 1) bit_count++
4. if (bit_count % 2 == 0)
5.    parity_bit = 0
6. else
7.    parity_bit = 1
```

```
2D-Parity(data, nsplits):
1. rows = nsplits
2. cols = data.size() / rows
3. for every row in data
4.    row_parity_bits.push_back(calc_parity_bit(row))
5. for every col in data
6.    col_parity_bits.push_back(calc_parity_bit(col))
```

2. Program

1D-Parity

main.cpp

```cpp
#include <iostream>
#include <string>
#include <vector>

#include "parity.hpp"

int main(int, char**) {

    std::vector<unsigned short> sender_data;
    std::vector<unsigned short> receiver_data;

    std::string input_line;
    std::cout << "Enter the Data from Sender : ";
    std::getline(std::cin, input_line);
    for (const char &c : input_line) {
        if (c != ' ') {
            sender_data.push_back(c-'0');
```


```
        }
    }
    char parity;
    std::cout << "Enter Parity (E/O) : ";
    std::cin >> parity;

    PARITY_OPTION parity_option = parity == 'e' ? EVEN : ODD;

    ParityData pdata(sender_data, parity_option);

    std::cout << "Parity Bit : " << pdata.get_parity_bit() << std::endl;

    std::cout << "Data Transmitted : " << input_line << pdata.get_parity_bit() <<
std::endl;

    std::cout << "Enter the Data from Receiver : ";
    std::cin.ignore();
    std::getline(std::cin, input_line);
    unsigned short parity_bit = input_line.back() - '0';
    input_line.pop_back();
    for (const char &c : input_line) {
        if (c != ' ') {
            receiver_data.push_back(c-'0');
        }
    }

    bool is_correct = pdata.check_with_received(receiver_data, parity_bit);

    is_correct ? std::cout << "CORRECT" : std::cout << "INVALID";

    std::cout << std::endl;
}
```

parity.cpp

```
#include "parity.hpp"

ParityData::ParityData(const std::vector<unsigned short>& data, PARITY_OPTION
parity_option = EVEN) : data(data) {
    this -> parity_option = parity_option;
    this -> parity_bit = ParityData::calculate_parity_bit(data, this -> parity_option);
}
```


```
unsigned short ParityData::calculate_parity_bit(const std::vector<unsigned short>& data,
PARITY_OPTION parity_option) {
    unsigned count = 0;

    for (auto e : data) {
        if (e == 1) {
            count++;
        }
    }

    bool is_count_even = count % 2 == 0;

    unsigned short parity_bit;

    switch(parity_option) {
        case EVEN:
            parity_bit = is_count_even == true ? 0 : 1;
            break;
        case ODD:
            parity_bit = is_count_even == true ? 1 : 0;
            break;
    }

    return parity_bit;
}

bool ParityData::check_with_received(const std::vector<unsigned short>& to_check, const
unsigned short& parity_bit) {
    unsigned short parity_bit_expected = ParityData::calculate_parity_bit(to_check, this
-> parity_option);
    bool is_correct = parity_bit == parity_bit_expected ? true : false;

    return is_correct;
}
```

parity.hpp

```
#pragma once

#include <vector>

enum PARITY_OPTION {
    EVEN,
```


```
    ODD
};

class ParityData {
    public:
        ParityData(const std::vector<unsigned short> &data, PARITY_OPTION option);
        bool check_with_received(const std::vector<unsigned short>& data, const unsigned
short& parity_bit);
        static unsigned short calculate_parity_bit(const std::vector<unsigned short>
&data, PARITY_OPTION option);
        unsigned short get_parity_bit() { return this -> parity_bit; };
    private:
        const std::vector<unsigned short> data;
        unsigned short parity_bit;
        PARITY_OPTION parity_option;
};
```

2D-Parity

main.cpp

```
#include <iostream>
#include <string>
#include <vector>

#include "parity.hpp"

template<typename T>
std::ostream& operator<<(std::ostream& os, const std::vector<T> &input) {
    for (auto const& i: input) {
        os << i << " ";
    }
    return os;
}

// 2D Parity Program
int main(int, char**) {

    std::vector<unsigned short> sender_data;
    std::vector<unsigned short> receiver_data;

    std::string input_line;
    std::cout << "Enter the Data from Sender : ";
    std::getline(std::cin, input_line);
    for (const char &c : input_line) {
```


```
        if (c != ' ') {
            sender_data.push_back(c-'0');
        }
    }
    char parity;
    std::cout << "Enter Parity (E/O) : ";
    std::cin >> parity;

    PARITY_OPTION parity_option = parity == 'e' ? EVEN : ODD;

    unsigned short splits;
    std::cout << "Enter how many splits you want : ";
    std::cin >> splits;

    // create the parity data object
    ParityData pdata(sender_data, splits, parity_option);

    std::cout << "Enter the Data from Receiver(row-wise) with parity bits : ";
    std::getline(std::cin, input_line);
    input_line.clear();
    std::getline(std::cin, input_line);
    for (const char &c : input_line) {
        if (c != ' ') {
            receiver_data.push_back(c-'0');
        }
    }

    // check if the data received is correct
    bool is_correct = pdata.check_with_received(receiver_data);

    is_correct ? std::cout << "CORRECT" : std::cout << "INVALID";

    std::cout << std::endl;
}

// OUTPUT
// Enter the Data from Sender : 1100
// Enter Parity (E/O) : e
// Enter how many splits you want : 2
// Enter the Data from Receiver with parity bits : 11000011
// CORRECT
```

parity.cpp

```cpp
#include "parity.hpp"

#include <iostream>

ParityData::ParityData(
    const std::vector<unsigned short>& data,
    unsigned short split,
    PARITY_OPTION parity_option = EVEN) {
        if (data.size() % split != 0) {
            throw "CANNOT SPLIT!";
        }
        this -> parity_option = parity_option;

        unsigned rows = split;
        unsigned cols = data.size() / split;
        this -> nrows = rows;
        this -> ncols = cols;

        // create our data matrix
        for (unsigned i = 0 ; i < rows ; i++) {
            std::vector<unsigned short> temp;
            for (unsigned j = 0 ; j < cols ; j++) {
                temp.push_back(data.at( i*cols+j ));
            }
            this -> data.push_back(temp);

            // meanwhile calculate the parity bit and push it as well
            unsigned short row_parity_bit = ParityData::calculate_parity_bit(temp,
parity_option);
            this -> row_parity_bits.push_back(row_parity_bit);
        }

        // calculate the column parity bits
        for (unsigned j = 0 ; j < cols ; j++) {
            std::vector<unsigned short> temp;
            for (unsigned i = 0 ; i < rows ; i++) {
                temp.push_back(this -> data.at(i).at(j));
            }

            unsigned short col_parity_bit = ParityData::calculate_parity_bit(temp,
parity_option);
            this -> column_parity_bits.push_back(col_parity_bit);
        }
```


```
        for (unsigned i = 0 ; i < rows ; i++) {
            for (unsigned j = 0 ; j < cols ; j++) {
                std::cout << this -> data.at(i).at(j);
            }
            std::cout << this -> row_parity_bits.at(i);
        }
        for (unsigned i = 0 ; i < this -> column_parity_bits.size() ; i++) {
            std::cout << this -> column_parity_bits.at(i);
        }
        std::cout << std::endl;

}

unsigned short ParityData::calculate_parity_bit(const std::vector<unsigned short>& data,
PARITY_OPTION parity_option) {
    unsigned count = 0;

    for (auto e : data) {
        if (e == 1) {
            count++;
        }
    }

    bool is_count_even = count % 2 == 0;

    unsigned short parity_bit;

    switch(parity_option) {
        case EVEN:
            parity_bit = is_count_even == true ? 0 : 1;
            break;
        case ODD:
            parity_bit = is_count_even == true ? 1 : 0;
            break;
    }

    return parity_bit;
}

bool ParityData::check_with_received(const std::vector<unsigned short>& to_check) {

    // check the row parity bits
    for (unsigned i = 0 ; i < this -> nrows ; i++) {
        unsigned parity_found = to_check.at((i * (this -> ncols + 1)) + this -> ncols);
```


```
        unsigned parity_expected = this -> row_parity_bits.at(i);

        if (parity_found != parity_expected) {
            return false;
        }
    }

    // check the column parity bits
    for (unsigned j = 0 ; j < this -> ncols ; j++) {
        // do the math okay ? i'm dumb i have no clue how i did this
        unsigned parity_found = to_check.at( (this -> nrows * (this -> ncols + 1)) + j);
        unsigned parity_expected = this -> column_parity_bits.at(j);

        if (parity_found != parity_expected) {
            return false;
        }
    }

    return true;
}
```

parity.hpp

```
#pragma once

#include <vector>

enum PARITY_OPTION {
    EVEN,
    ODD
};

class ParityData {
    public:
        ParityData(const std::vector<unsigned short> &data,  unsigned short split,
PARITY_OPTION option);
        bool check_with_received(const std::vector<unsigned short>& data);
        static unsigned short calculate_parity_bit(const std::vector<unsigned short>
&data, PARITY_OPTION option);
    private:
        std::vector<std::vector<unsigned short>> data;
        std::vector<unsigned short> row_parity_bits;
        std::vector<unsigned short> column_parity_bits;
```


```
        unsigned nrows, ncols;
        PARITY_OPTION parity_option;
};
```

3. Results

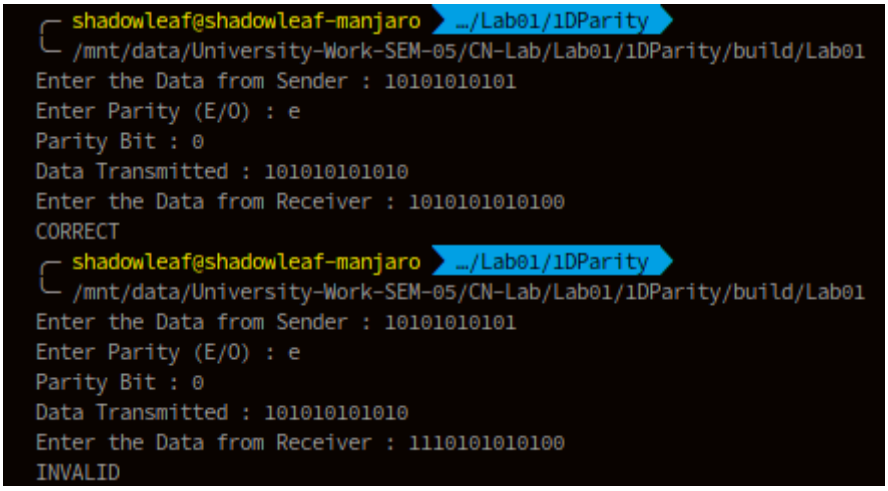


Figure 0-1 1D-Parity

Here the data taken is of 11 bits, the corresponding parity bit is calculated, the receiver then gets the data and checks with the parity bit, here we've taken two examples, in example 1 the data is intact, hence the output is CORRECT, while in the next example the data is mutated and the message outputs INVALID.

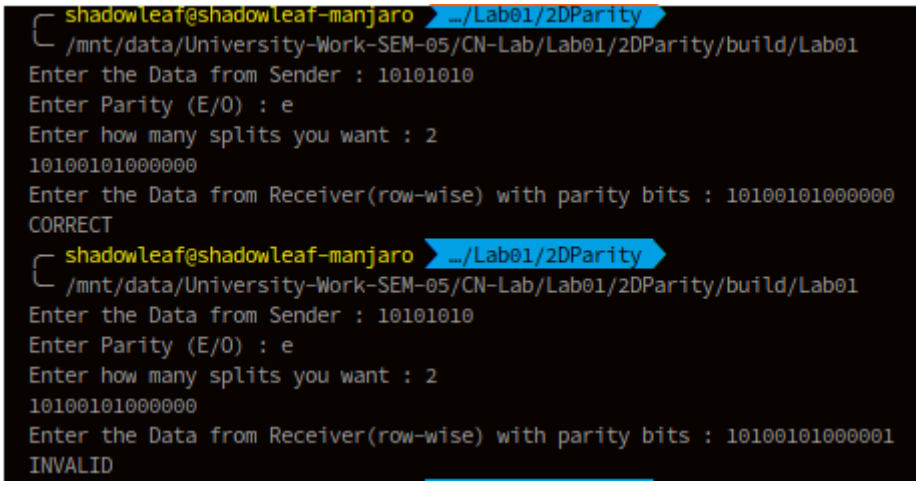


Figure 0-2 2D-Parity

In 2D parity, the data is given and also the number of splits is asked from the user, the data is correspondingly split and the row, column parity bits are calculated and stored in a vector, the complete data along with the parity bits are displayed, this data is transmitted, on the receiver end the data is taken and the row, column parity bits are checked, if they are correct then CORRECT is displayed, else INVALID, in the above output, the first output shows when the receiver gets the intact data and in the latter part the data is altered.

4. Analysis and Discussions

- Why can this method not be used to correct errors?

Parity Bit for error detection is prone to data mutations, such as in 1D-Parity is even number of bits are simultaneously mutated then the parity bits remain unchanged, the receiver will not be able to detect the error in the data. Same goes to 2D-Parity where if the bits in the data grid are mutated in a square region the receiver will not be able to detect corrupted data. Another problem with this method is that the parity bit itself can mutate, which will represent the data to be corrupted even though the data transmitted might be correct.

- How are 1D and 2D parity different?

1D Parity has only one parity bit, while 2D parity has multiple bits based on how the data is divided, 2D parity can detect burst errors, while 1D can only detect single bit errors or odd-number of bit errors.

- What are the limitations of this method of error detection?

Parity bits cannot detect all forms of bit error

For 2D Parity, rows + column number of parity bits are added that are to be transmitted along with data, more data, means that there's more chances of corruption.

5. Conclusions

Parity bits are a very simple way to check if the transmitted data is corrupted on the way, although it cannot detect all forms of errors, it is very simple to implement.

6. Comments

   a. Limitations of the experiment

The experiment assumes the data to be binary values, although in the real world the data is taken as bit streams, the experiment is limited to proof of concept.

b. Limitations of the results obtained

The results obtained can be checked for bitstrings, and does not work on generic data.

c. Learning

The concept of Parity bits (1D and 2D) was learnt in this lab.

d. Recommendations

Instead of using an array/vector to represent the data, the data should be taken as byte streams and encoded accordingly, such as a string, or a binary file, this was the program will be more generic.

Experiment 2: Error Detection using CRC-CCITT

Aim: To apply CRC (CCITT Polynomial) for error detection

Objective: After carrying out this experiment, students will be able to:

- Apply CRC CCITT to develop codes for error detection
- Analyse how this CRC is able to detect bit errors irrespective of their length and position in the data

Problem statement: You are required to write a program that uses CRC to detect burst errors in transmitted data. Initially, write the program using the CRC example you studied in class. Your final program should ask the user to input data and choose a generator polynomial from the list given in the figure below. Your program is required to calculate the checksum and the transmitted data. Subsequently, the user enters the received data. Applying the same generator polynomial on the received data should result in a remainder of 0.

Name	Polynomial	Application
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

Analysis: While analyzing your program, you are required to address the following points:

- How is this method different from 2D parity scheme that you have implemented previously?
- What are the limitations of this method of error detection?

MARKS DISTRIBUTION

Component	Maximum Marks	Marks Obtained
Preparation of Document	7	
Results	7	
Viva	6	
Total	**20**	

Submitted by: Satyajit Ghana

Register No: 17ETCS002159

7. Algorithm/Flowchart

calc_crc(message, crc_poly)

```
1. remainder = 0x0
2. for (every byte in message)
3.       remainder = byte << (crc_poly.length - 8)
4.       for i = 0 to 7
5.       if (TOP_BIT(remainder) == 1)
6.                   remainder = (remainder << 1) XOR crc_poly
7.             else remainder = remainder << 1
```

check_crc(message, crc_poly)

```
1. crc = calc_crc(message, crc_poly)
2. if any bit of crc is ON return false
3. else return true
```

8. Program

main.c

```
#include <iostream>
#include <bitset>
#include <vector>

#include "crc_lib.hpp"

/**
 * Implement :
 * CRC-8
 * CRC-10
 * CRC-16
 * CRC-32
 */

std::ostream& operator<<(std::ostream& out, std::vector<std::bitset<8>> message) {
    for (auto& byte : message) {
        out << byte << " ";
    }
```


```
    return out;
}

int main(int, char**) {

    std::cout << "Enter your message (in binary) : ";
    std::string input;
    std::getline(std::cin, input);

    // pad extra zero to make the number of bits a multiple of 8
    unsigned to_pad = input.size() % 8 != 0 ? (8 - input.size() % 8) : 0;
    for (int i = 0 ; i < to_pad ; i++) {
        input = "0"+input;
    }

    std::vector<std::bitset<8>> message;

    for (int i = 0 ; i+8 <= input.size() ; i += 8) {
        message.push_back(std::bitset<8>(input.substr(i, 8)));
    }

    std::cout << "MESSAGE : " << message << std::endl;

    CRC_OPTION crc_option;

retake_choice:
    std::cout <<
        "Select a CRC Polynomial : \n"
        "1.\tCRC8\n"
        "2.\tCRC10\n"
        "3.\tCRC16\n"
        "4.\tCRC32\n"
        "Your Choice : ";
    char choice;
    std::cin >> choice;

    switch(choice) {
        case '1':
            crc_option = CRC8;
            break;
        case '2':
            crc_option = CRC10;
            break;
        case '3':
            crc_option = CRC16;
```


```
            break;
        case '4':
            crc_option = CRC32;
            break;
        default:
            std::cout << "Wrong Choice !";
            goto retake_choice;
    }

    std::tuple<std::bitset<32>, unsigned> crc_calculated = CRC::calc_crc(message, crc_opt
ion);

    std::cout << "CRC : " << std::get<0>(crc_calculated).to_string().substr(32-
std::get<1>(crc_calculated)) << std::endl;

    std::cin.ignore();
    std::cout << "Enter the data received (with CRC padded at the end) : ";
    std::getline(std::cin, input);

    to_pad = input.size() % 8 != 0 ? (8 - input.size() % 8) : 0;
    for (int i = 0 ; i < to_pad ; i++) {
        input = "0"+input;
    }

    message.clear();
    for (int i = 0 ; i+8 <= input.size() ; i += 8) {
        message.push_back(std::bitset<8>(input.substr(i, 8)));
    }

    std::cout << "MESSAGE : " << message << std::endl;

    CRC::check_crc(message, crc_option) ? std::cout << "PASS" : std::cout << "FAIL";

    std::cout << std::endl;
}
```

crc_lib.hpp

```
#pragma once

#include <bitset>
#include <vector>
#include <tuple>
```


```
enum CRC_OPTION {
    CRC8, CRC10, CRC16, CRC32
};

class CRC {
public:
    static std::tuple<std::bitset<32>, unsigned> calc_crc(std::vector<std::bitset<8>> &me
ssage, CRC_OPTION crc_option);
    static std::tuple<std::bitset<32>, unsigned> get_crc_polynomial(CRC_OPTION CRC_OPTION
);
    static bool check_crc(std::vector<std::bitset<8>>& message, CRC_OPTION crc_option);
};
```

crc_lib.cpp

```
#include "crc_lib.hpp"

#include <iostream>
#include <tuple>

// returns the CRC polynomial with the CRC polynomial length
std::tuple<std::bitset<32>, unsigned> CRC::get_crc_polynomial(CRC_OPTION crc_option) {

    std::bitset<32> crc_poly;
    unsigned crc_poly_len;

    switch(crc_option) {
        case CRC8:
            crc_poly = std::bitset<32>(0xD5);
            crc_poly_len = 8;
            break;
        case CRC10:
            crc_poly = std::bitset<32>(0x233);
            crc_poly_len = 10;
            break;
        case CRC16:
            crc_poly = std::bitset<32>(0x1021);
            crc_poly_len = 16;
            break;
        case CRC32:
            crc_poly = std::bitset<32>(0x04C11DB7);
            crc_poly_len = 32;
```


```
            break;
    }

    return std::make_tuple(crc_poly, crc_poly_len);
}

std::tuple<std::bitset<32>, unsigned> CRC::calc_crc(std::vector<std::bitset<8>>& message,
 CRC_OPTION crc_option) {
    // Fetch the CRC polynomial
    std::tuple<std::bitset<32>, unsigned> _crc_poly = CRC::get_crc_polynomial(crc_option)
;

    std::bitset<32> crc_poly = std::get<0>(_crc_poly);
    unsigned crc_poly_len = std::get<1>(_crc_poly);

    std::cout << "POLYNOMIAL : " << crc_poly << std::endl;
    // initialize the remainder with 0
    std::bitset<32> remainder(0x0);

    // Perform for every byte in message
    for (auto& byte : message) {
        // load the byte to the remainder
        remainder ^= ( (std::bitset<32>(byte.to_string())) << (crc_poly_len - byte.size()
) );

        // perform for every bit in the byte
        for (unsigned i = 0 ; i < 8 ; i++) {
            // std::cout << "REM : " << remainder << std::endl;
            // check if the top bit is 1
            if (remainder.test(crc_poly_len-1)) {
                remainder = (remainder << 1) xor crc_poly;
            } else {
                remainder = remainder << 1;
            }
        }
    }

    // std::cout << "CRC : " << remainder << std::endl;

    return std::make_tuple(remainder, crc_poly_len);
}

bool CRC::check_crc(std::vector<std::bitset<8>>& message, CRC_OPTION crc_option) {
    std::tuple<std::bitset<32>, unsigned> crc = CRC::calc_crc(message, crc_option);
```


```
    std::bitset<32>val(std::get<0>(crc).to_string().substr(32-std::get<1>(crc)));

    if (val.any()) {
        return false;
    } else {
        return true;
    }
}
```

9. Results

```
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02
└─ /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02/build/Lab02
Enter your message (in binary) : 10101010
MESSAGE : 10101010
Select a CRC Polynomial :
1.      CRC8
2.      CRC10
3.      CRC16
4.      CRC32
Your Choice : 1
POLYNOMIAL : 00000000000000000000000011010101
CRC : 00011101
Enter the data received (with CRC padded at the end) : 1010101000011101
MESSAGE : 10101010 00011101
POLYNOMIAL : 00000000000000000000000011010101
PASS
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02
└─ /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02/build/Lab02
Enter your message (in binary) : 10101010
MESSAGE : 10101010
Select a CRC Polynomial :
1.      CRC8
2.      CRC10
3.      CRC16
4.      CRC32
Your Choice : 1
POLYNOMIAL : 00000000000000000000000011010101
CRC : 00011101
Enter the data received (with CRC padded at the end) : 1101010100011100
MESSAGE : 11010101 00011100
POLYNOMIAL : 00000000000000000000000011010101
FAIL
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02
└─
```

Figure 0-1 OUTPUT 1

Explanation for Figure 0-1 OUTPUT 1:

Here the data bits are taken as 10101010, and the CRC chosen was CRC8 which is 111010101, performing manual CRC calculation resulted in the CRC to be 00011101, which is same as the output from the program, the message is then padded with the CRC at the end and the received

message is given to the program which then verifies the output. Printing PASS if the message is error free, FAIL otherwise. Following the output, is the input for the same message but the message that is received is altered, here the error is detected since the remainder from the message is non-zero.

```
shadowleaf@SHADOWLEAF-ROG /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02/build/Lab02
Enter your message (in binary) : 1010
MESSAGE : 00001010
Select a CRC Polynomial :
1.      CRC8
2.      CRC10
3.      CRC16
4.      CRC32
Your Choice : 1
POLYNOMIAL : 00000000000000000000000011010101
CRC : 01010110
Enter the data received (with CRC padded at the end) : 11101010100
MESSAGE : 00000111 01010100
POLYNOMIAL : 00000000000000000000000011010101
PASS
shadowleaf@SHADOWLEAF-ROG /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02
```

Figure 0-2 OUTPUT 2

Explanation for Figure 0-2 OUTPUT 2:

Here the data taken is 1010, and the respective CRC is calculated, assuming the data received by the receiver is mutated and now the data is a factor of the CRC polynomial, here the error in the data is not detected, which is one of the drawbacks of CRC.

10. Analysis and Discussions

The conventional method also generally utilizes cyclic redundancy check (CRC) bits for error detection purposes. In particular, a fixed number of CRC bits are appended to the end of each message block and have a predetermined relationship with the corresponding message block. The receiver receives both the message block and the CRC bits following that message block, and tries to re-establish the relationship therebetween. If the relationship is satisfied, the message block is considered without error. Otherwise, an error has occurred during the transmission of that block. This method is further explained in greater detail below.

First, a CRC generating polynomial, $g_l(x)$, of order 1, is chosen. A common way of choosing the CRC generating polynomial is that $g_l(x)$ should satisfy $gcd(g_l(x), x) = 1$ for each and every i between 0 and 1, inclusive, wherein 1 and i are integers, and the function $gcd(A(X).B(x))$ is defined as the greatest common divider of polynomials A(X) and B(x). Examples of suitable g(x) include $g_4(x) = x^4 + x^3 + x^2 + x + 1$ for l=4; $g_7(x) = x^7 + x^6 + x^4 + 1$ for l=7:$g_8(x) = x^8 + x^7 + x^4 + x^3 + x + 1$ for l=8; and $g(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1$ for l=12. The information of CRC generating polynomial is stored in both the transmitter and the receiver. **(Shien S.L, 2007)**

Shien, S.L., Industrial Technology Research Institute, 2007. *Cyclic redundancy check modification for message length detection and error detection*. U.S. Patent 7,240,273.

2D parity implemented in the previous lab is prone to leave errors made in a square pattern undetected, a large part of the burst errors can be detected using this method, although still another huge segment of the burst errors go undetected, this is where the CRC comes in, the number of bits in CRC is fixed, i.e. the number of redundant bits are fixed $O(1)$, unlike where in 2D parity the number of bits increase $O(n)$ as the message length increases, in terms of space complexity CRC is better.

Albeit there are a few limitations associated with CRC, such as:

(i) The error in the data can do undetected if the message is mutated such that it is a factor of the CRC polynomial itself, as shown in Figure 0-2 OUTPUT 2.

(ii) Because a CRC is based on division, no polynomial can detect errors consisting of a string of zeroes prepended to the data, or of missing leading zeroes.

(iii) Single bit errors will be detected by any polynomial with at least two terms with non-zero coefficients.

(iv) Burst errors of length n will be detected by any polynomial of degree n or greater which has a non-zero x^0 term.

11. Conclusions

CRC is a much better upgrade to the 2D parity methods, with very low probability of errors left undetected by the receiver. It is sweet, short, although takes a little amount of computation, which can be fixed by using a lookup table, works great and is used practically in Networking.

12. Comments

a. Limitations of the experiment

The implementation part of the program is inefficient, i.e. it calculates the CRC for every byte in the data, although this is not required, since a lookup table can be created that will solve the problem in $O(1)$ time for every bit, and $O(n)$ in total, where n is the number of bytes in the message.

b. Limitations of the results obtained

The results are checked and the limitation of the CRC is shown, other than that and a few other limitations due to the maths behind CRC, no other limitations are known.

c. Learning

Through this lab the concept of CRC was learnt that is used for error detection.

d. Recommendations

Make the implementation more generic and more efficient, the program should work for any sort of data, files, strings, everything, this can be achieved by treating the data as a character array of bytes, then each byte can be processed individually.

Experiment 3: Neighbour Table Determination

Aim: To create neighbor table for a given network topology

Objective: After carrying out this experiment, students will be able to:

- Generate neighbor table for all the nodes in a given topology.
- Analyse how this is useful in the process of routing data

Problem statement: You are required to write a program that calculates neighbor table for all the nodes in a given network. Consider a network with 10 nodes that is deployed in an area of 500 m^2. Your program should initially determine the distance between each node and all other nodes. Then the range of the nodes is given as input to the user. Using this range information, determine the neighbors of all the nodes.

Analysis: While analyzing your program, you are required to address the following points:

- How this is useful in the process of routing data?
- For a 3D topology, how would your program need to be changed?

MARKS DISTRIBUTION

Component	Maximum Marks	Marks Obtained
Preparation of Document	7	
Results	7	
Viva	6	
Total	**20**	

Submitted by: Satyajit Ghana

Register No: 17ETCS002159

13. Algorithm/Flowchart

generate_random_nodes(length, breadth):

1. nodes_pos.push_back({random_int(0, length), random_int(0, breadth)});

calculate_neighbour_table():

1. for i = 0 to nnodes - 1
2. for j = 0 to nnodes - 1
3. if dist = calc_dist(node_pos.at(i), node_pos.at(j) < node_range.at(i)
4. neighbor_table.add_edge(i, j, dist)

calc_dist(pos1, pos2):

1. (post1.x - pos2.x)^2 + (pos1.y - pos2.y)^2

14. Program

main.cpp

```cpp
#include <iostream>

#include "network.hpp"

int main(int, char**) {
    using namespace std;

    Network mynet(10);

    // Generate the Random Nodes
    mynet.generate_random_nodes_with_dimensions(10, 50);

    mynet.print_nodes_pos();

    vector<int> ranges(10);
    cout << "Enter the Ranges for the 10 Nodes : ";
    for (int i = 0; i < 10; i++)
        cin >> ranges[i];

    // Set the Node Ranges given by the user and calculate the neighbour table
```


```
    mynet.set_node_ranges(ranges);

    mynet.print_neighbour_table();
}
```

network.cpp

```
#include "network.hpp"

#include <iostream>
#include <random>

Network::Network(int n_nodes) : n_nodes(n_nodes), neighbour_table(n_nodes), node_ranges(n
_nodes), nodes_pos(n_nodes) {}

void Network::generate_random_nodes_with_dimensions(int length, int breadth) {
    std::random_device random_device;
    std::mt19937 random_engine(random_device());
    std::uniform_int_distribution<int> l_dist(0, length);
    std::uniform_int_distribution<int> b_dist(0, breadth);

    this->nodes_pos.clear();

    for (int i = 0; i < this->n_nodes; i++) {
        nodes_pos.push_back({l_dist(random_engine), b_dist(random_engine)});
    }

    std::cout << this->n_nodes << " Random Nodes Generated" << std::endl;
}

void Network::print_nodes_pos() {
    int i = 0;
    for (auto& node : this->nodes_pos) {
        std::cout << "NODE " << i << " : (" << node.first << ", " << node.second << ")" <
< std::endl;
        i++;
    }
}

void Network::set_node_ranges(std::vector<int>& ranges) {
    if (ranges.size() != this->nodes_pos.size()) {
        throw "ranges length does not match nodes_pos length";
    }
```


```
    this->node_ranges.clear();

    this->node_ranges.assign(ranges.begin(), ranges.end());

    this->gen_fully_connected_network(this->node_ranges);
}

template <typename T>
double Network::calc_dist(std::pair<T, T> pos1, std::pair<T, T> pos2) {
    return std::sqrt(std::abs(
        ((pos1.first - pos2.first) * (pos1.first - pos2.first) + (pos1.second - pos2.seco
nd) * (pos1.second - pos2.second))));
}

// Generates the network from the given ranges
void Network::gen_fully_connected_network(std::vector<int>& ranges) {
    if (this->node_ranges.size() != this->nodes_pos.size()) {
        throw "WTF did you do ? and HTF did you do that ? i'm not even public ";
    }

    this->neighbour_table.clear_graph();

    for (int node_idx = 0; node_idx < this->nodes_pos.size(); node_idx++) {
        for (int node_jdx = 0; node_jdx < this->nodes_pos.size(); node_jdx++) {
            double dist = Network::calc_dist(this->nodes_pos.at(node_idx), this-
>nodes_pos.at(node_jdx));
            if (dist <= node_ranges.at(node_idx)) {
                this->neighbour_table.add_edge(node_idx, node_jdx, dist);
            }
        }
    }
}

void Network::print_neighbour_table() {
    if (this->node_ranges.empty()) {
        std::cout << "EMPTY" << std::endl;
        return;
    }

    this->neighbour_table.print_graph();
}
```


network.hpp

```cpp
#pragma once

#include <math.h>
#include <vector>

#include "graph.hpp"

class Network {
   public:
    Network(int nodes);
    int n_nodes;
    std::vector<std::pair<int, int>> nodes_pos;
    std::vector<int> node_ranges;
    Graph neighbour_table;

    void set_node_ranges(std::vector<int>& ranges);
    void generate_random_nodes_with_dimensions(int length, int breadth);
    void print_nodes_pos();
    void print_neighbour_table();

    template <typename T>
    static double calc_dist(std::pair<T, T>, std::pair<T, T>);

   private:
    void gen_fully_connected_network(std::vector<int>& ranges);
};
```

graph.cpp

```cpp
#include "graph.hpp"

#include <iostream>

Graph::Graph(int nodes) : adj_list(nodes), order(nodes) {}

void Graph::add_edge(int src, int dest, double weight) {
    if (src >= adj_list.size() || dest >= adj_list.size()) {
        throw "Tried to Add Edge for Vertex that does not exist";
    }

    adj_list.at(src).push_back({dest, weight});
}

void Graph::print_graph() {
```


```
    std::cout << "GRAPH" << std::endl;
    int i = 0;
    for (auto& row : adj_list) {
        std::cout << "NODE " << i << " -> ";
        for (auto& ele : row) {
            std::cout << ele.first << " : " << ele.second << " , ";
        }
        std::cout << std::endl;
        i++;
    }
}
```

graph.hpp

```
#pragma once

#include <vector>

// Weighted DiGraph using Adjacency List
class Graph {
   public:
    Graph(int);
    std::vector<std::vector<std::pair<int, double>>> adj_list;

    const int order;

    void clear_graph() {
        this->adj_list.clear();
        this->adj_list.resize(order);
    };
    void add_edge(int src, int dest, double weight);

    bool is_empty() { return adj_list.empty(); };

    void print_graph();
};
```


15. Results

```
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab03
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab03/build/Lab03
10 Random Nodes Generated
NODE 0 : (9, 36)
NODE 1 : (1, 20)
NODE 2 : (2, 42)
NODE 3 : (4, 50)
NODE 4 : (9, 13)
NODE 5 : (7, 46)
NODE 6 : (10, 4)
NODE 7 : (2, 42)
NODE 8 : (5, 47)
NODE 9 : (0, 36)
Enter the Ranges for the 10 Nodes : 10 5 12 13 5 8 19 20 10 13
GRAPH
NODE 0 -> 0 : 0 , 2 : 9.21954 , 7 : 9.21954 , 9 : 9 ,
NODE 1 -> 1 : 0 ,
NODE 2 -> 0 : 9.21954 , 2 : 0 , 3 : 8.24621 , 5 : 6.40312 , 7 : 0 , 8 : 5.83095 , 9 : 6.32456 ,
NODE 3 -> 2 : 8.24621 , 3 : 0 , 5 : 5 , 7 : 8.24621 , 8 : 3.16228 ,
NODE 4 -> 4 : 0 ,
NODE 5 -> 2 : 6.40312 , 3 : 5 , 5 : 0 , 7 : 6.40312 , 8 : 2.23607 ,
NODE 6 -> 1 : 18.3576 , 4 : 9.05539 , 6 : 0 ,
NODE 7 -> 0 : 9.21954 , 2 : 0 , 3 : 8.24621 , 5 : 6.40312 , 7 : 0 , 8 : 5.83095 , 9 : 6.32456 ,
NODE 8 -> 2 : 5.83095 , 3 : 3.16228 , 5 : 2.23607 , 7 : 5.83095 , 8 : 0 ,
NODE 9 -> 0 : 9 , 2 : 6.32456 , 5 : 12.2066 , 7 : 6.32456 , 8 : 12.083 , 9 : 0 ,
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab03
```

Figure 0-1 Output

In Figure 0-1 Output 10 random nodes are generated in an area of 500 square units using a standard normal distribution, then the range of each of those nodes is taken as input from the user.

Since this is a fully connected network, we traverse breadth first, across all connection, on each connection, the distance from the source to the destination is calculated, and if the range of the source node is greater than or equal to the range, then this connection is marked as valid, a matrix of such valid connection is made, this is the neighbor table for the network.

16. Analysis and Discussions

- How this is useful in the process of routing data?

Neighbour Table is useful since from the table, every node knows, which node can reach which other nodes in the network. Suppose a packet needs to be sent from a source to a destination

node, the source node can calculate the shortest path the packet has to travel, it can also determine whether the packet will even reach the destination. This will optimize the bandwidth of the network and utilize the resources better. Since we know the neighbors and their destination, the route is predetermined, this is an added advantage for the network.

- For a 3D topology, how would your program need to be changed?

A 3D topology of network can be flattened to 2D network as well, it's just a matter of perception, the program would work even for the 3D network, the only change that has to be made is the dimensionality for the network, the position vectors of the nodes is now $\mathbb{R}^3$, the Euclidian distance formula will have to be changed, other than that everything is same, the Adjacency matrix is 2D and can very well store the connections for the 3D topology.

17. Conclusions

Neighbour table is a pretty simple mechanism by which the nodes can determine the paths in which the packet has to be travelled, the fact that the path is predetermined is the main advantage in this mechanism, one big disadvantage is that the neighbor table has to be determined again entirely if any node is damaged or goes offline. The complexity of the algorithm is $O(n^2)$. The time will increase polynomially.

18. Comments

a. Limitations of the experiment

The complexity increases as the number of nodes increase.

b. Limitations of the results obtained

The results are obtained for a very small set of nodes, which fails to depict the limitation of this experiment.

c. Learning

We learnt a mechanism for generating a Neighbour table for a given topology of network.

d. Recommendations

The program needs to be simulated for a larger number of nodes, with 2D and 3D topology as well.

Experiment 4: Distance Vector Routing

Aim: To generate routing tables for a network of routers using Distance Vector Routing

Objective: After carrying out this experiment, students will be able to:

- Generate routing tables for a given network using Distance Vector Routing
- Analyze the reasons why Distance Vector Routing is adaptive in nature

Problem statement: You are required to write a program that can generate routing tables for a network of routers. Take the number of nodes and the adjacency matrix as input from user. Your program should use this adjacency matrix and create routing tables for all the nodes in the network. The routing table should consist of one entry per destination. This entry should contain the total cost and the outgoing line to reach that destination.

Analysis: While analyzing your program, you are required to address the following points:

- Why is Distance Vector Routing classified as an adaptive routing algorithm?
- Limitations of Distance Vector Routing

MARKS DISTRIBUTION

Component	Maximum Marks	Marks Obtained
Preparation of Document	7	
Results	7	
Viva	6	
Total	**20**	

Submitted by: SATYAJIT GHANA

Register No: 17ETCS002159

19. Algorithm/Flowchart

Input: Graph and a source vertex src

Output: Shortest distance to all vertices from src. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

1) This step initializes distances from source to all vertices as infinite and distance to source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.

2) This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.
a) Do following for each edge u-v

If dist[v] > dist[u] + weight of edge uv, then update dist[v]

dist[v] = dist[u] + weight of edge uv

3) This step reports if there is a negative weight cycle in graph. Do following for each edge u-v

If dist[v] > dist[u] + weight of edge uv, then "Graph contains negative weight cycle"

The idea of step 3 is, step 2 guarantees shortest distances if graph doesn't contain negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle

20. Program

main.cpp

```cpp
#include <iostream>

#include "routing_table.hpp"

// Distance Vector Routing
int main(int, char**) {
    using namespace std;
    int order;
    cout << "Enter the Order of the Matrix : ";
    cin >> order;
    cout << "Enter the Adjacency Matrix : " << endl;

    vector<vector<int>> adj_mat;

    for (int i = 0 ; i < order ; i++) {
        std::vector<int> row(order);
        for (int j = 0 ; j < order ; j++) {
            cin >> row[j];
        }
        adj_mat.push_back(row);
    }

    // create the routing table
    RoutingTable routing_table(adj_mat);

    // calculate the shortest paths
    routing_table.apply_bellman_ford();

    // print the table
    routing_table.print_routing_table();

}
```

graph.hpp

```cpp
#pragma once

#include <vector>

// Weighted DiGraph using Adjacency List
class Graph {
```


```
    public:
        Graph(int);
        std::vector<std::vector<std::pair<int, double>>> adj_list;
        std::vector<std::vector<int>> adj_matrix;

        const int order;

        void clear_graph();
        void add_edge(int src, int dest, double weight);

        bool is_empty() { return adj_list.empty(); };

        void print_graph();

};
```

graph.cpp

```
#include "graph.hpp"

#include <iostream>

Graph::Graph(int nodes) : adj_list(nodes), order(nodes), adj_matrix(nodes) {
    for (auto& row : adj_matrix) {
        row.resize(nodes);
        for (auto& node : row) {
            node = 0;
        }
    }
}

void Graph::add_edge(int src, int dest, double weight) {
    if (src >= adj_list.size() || dest >= adj_list.size()) {
        throw "Tried to Add Edge for Vertex that does not exist";
    }

    adj_list.at(src).push_back({dest, weight});

    adj_matrix.at(src).at(dest) = weight;
}

void Graph::clear_graph() {
    this -> adj_list.clear(); this -> adj_list.resize(order);
    for (auto& row : adj_matrix) {
        for (auto& node : row) {
```


```
            node = 0;
        }
    }
};

void Graph::print_graph() {
    std::cout << "GRAPH" << std::endl;
    int i = 0;
    for (auto& row : adj_list) {
        std::cout << "NODE " << i << " -> ";
        for (auto& ele : row) {
            std::cout << ele.first << " : " << ele.second << " , ";
        }
        std::cout << std::endl;
        i++;
    }
}
```

routing_table.hpp

```
#pragma once

#include "graph.hpp"

class RoutingTable {
    public:
        RoutingTable(int nodes);
        RoutingTable(std::vector<std::vector<int>>& adj_matrix);
        Graph nodes;

        std::vector<std::vector<int>> dist_matrix;

        void apply_bellman_ford();
        void print_routing_table();
};
```

routing_table.cpp

```
#include "routing_table.hpp"

#include <limits>
#include <iostream>

RoutingTable::RoutingTable(int nodes) : nodes(nodes), dist_matrix(nodes) {
    // Initialize the table to 0
```


```
    for (int i = 0 ; i < nodes ; i++) {
        for (int j = 0 ; j < nodes ; j++) {
            dist_matrix.at(i).push_back(0);
        }
    }
}

RoutingTable::RoutingTable(std::vector<std::vector<int>>& adj_matrix) : RoutingTable(adj_
matrix.size()) {
    std::cout << adj_matrix.size() << std::endl;

    for (int i = 0 ; i < this -> nodes.order ; i++) {
        for (int j = 0 ; j < this -> nodes.order ; j++) {
            if (adj_matrix.at(i).at(j) > 0) {
                this -> nodes.add_edge(i, j, adj_matrix.at(i).at(j));
            }
        }
    }
}

void RoutingTable::print_routing_table() {
    int i = 0;
    for (auto& row : this -> dist_matrix) {
        std::cout << "SRC NODE " <<  i << " TO -> [ ";
        int j = 0;
        for (auto& node : row) {
            std::cout << j << " : " << node << " , ";
            j++;
        }
        std::cout << " ]" << std::endl;
        i++;
    }
}

void RoutingTable::apply_bellman_ford() {
    dist_matrix.clear();
    dist_matrix.resize(this -> nodes.order);

    for (int src = 0 ; src < this -> dist_matrix.size() ; src++) {

        auto& curr_dist = dist_matrix.at(src);

        curr_dist.resize(this -> nodes.order);

        // set distance from src to every destination as infinity
```


```
        for (auto& node : dist_matrix.at(src)) {
            node = std::numeric_limits<int>::max();
        }
        // distance from source to source is 0
        curr_dist.at(src) = 0;

        // relax all edges V-1 times
        for (int k = 0 ; k <= this -> nodes.order - 1 ; k++) {
            for (int i = 0 ; i < this -> nodes.order ; i++) {
                for (int j = 0 ; j < this -> nodes.order ; j++) {
                    if (this -
> nodes.adj_matrix.at(i).at(j) != 0 && curr_dist.at(i) != std::numeric_limits<int>::max()
) {
                        int weight = this -> nodes.adj_matrix.at(i).at(j);
                        curr_dist.at(j) =
                            std::min(
                                curr_dist.at(i) + weight,
                                curr_dist.at(j)
                                );
                    }
                }
            }
        }

    }
}
```

21. Results

```
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab04
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab04/build/Lab04
Enter the Order of the Matrix : 5
Enter the Adjacency Matrix :
0 6 0 7 0
0 0 5 8 4
0 2 0 0 0
0 0 3 0 9
2 0 7 0 0
5
SRC NODE 0 TO -> [ 0 : 0 , 1 : 6 , 2 : 10 , 3 : 7 , 4 : 10 ,  ]
SRC NODE 1 TO -> [ 0 : 6 , 1 : 0 , 2 : 5 , 3 : 8 , 4 : 4 ,  ]
SRC NODE 2 TO -> [ 0 : 8 , 1 : 2 , 2 : 0 , 3 : 10 , 4 : 6 ,  ]
SRC NODE 3 TO -> [ 0 : 11 , 1 : 5 , 2 : 3 , 3 : 0 , 4 : 9 ,  ]
SRC NODE 4 TO -> [ 0 : 2 , 1 : 8 , 2 : 7 , 3 : 9 , 4 : 0 ,  ]
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab04
```

Figure 0-1 OUTPUT

The Output here is the distance from each of the node to every other node, in form of a matrix, each of each node will store its own distance table

22. Analysis and Discussions

Distance Vector Routing is a dynamic routing algorithm in which each router computes distance between itself and each possible destination i.e. its immediate neighbors. Routers that use other adaptive protocols, such as grouped adaptive routing, find a group of all the links that could be used to get the packet one hop closer to its final destination. The router sends the packet out any link of that group which is idle. The link aggregation of that group of links effectively becomes a single high-bandwidth connection.

Disadvantages of Distance Vector routing –

- It is slower to converge than link state.
- It is at risk from the count-to-infinity problem.
- It creates more traffic than link state since a hop count change must be propagated to all routers and processed on each router. Hop count updates take place on a periodic basis, even if there are no changes in the network topology, so bandwidth-wasting broadcasts still occur.
- For larger networks, distance vector routing results in larger routing tables than link state since each router must know about all other routers. This can also lead to congestion on WAN links.

Note – Distance Vector routing uses UDP(User datagram protocol) for transportation.

23. Conclusions

A Note about Dynamic Routing

Routers that use some adaptive protocols, such as the Spanning Tree Protocol, in order to "avoid bridge loops and routing loops", calculate a tree that indicates the one "best" link for a packet to get to its destination. Alternate "redundant" links not on the tree are temporarily disabled—until

one of the links on the main tree fails, and the routers calculate a new tree using those links to route around the broken link.

24. Comments

a. Limitations of the experiment

Since DVR uses UDP, not all the packets reach the destination, the complexity hence is greater than O(n^3).

b. Limitations of the results obtained

The Routing Table here is pretty small, which does not justify the slowness of the algorithm, the program can be simulated for larger number of nodes and the time can be compared with other algorithms.

c. Learning

We learnt the algorithm of DVR, and its limitations from the implementation of the program, since it has O(n^3) complexity.

d. Recommendations

The node connections can be taken as a adjacency list instead of a matrix, since the matrix will always be larger than the adj-list.

Experiment 5: ARQ Mechanisms in DLL

Aim: To implement receiver algorithms for the different ARQ mechanisms at the Data Link Layer

Objective: After carrying out this experiment, students will be able to:

- implement receiver algorithms for the different ARQ mechanisms at the Data Link Layer
- Analyze the differences between the ARQ mechanisms

Problem statement: You are required to write a program that can receive frames at the data link layer. Assume that the user is entering the frames as the transmitter. You are required to implement stop and wait, go back N and selective repeat ARQ mechanisms. Consider that you have to transmit and receive a total of 20 frames using $W_T=W_R=1$, $W_T=5$ and $W_R=1$ and $W_T=W_R=5$ for stop and wait, go back N and selective repeat respectively

Analysis: While analyzing your program, you are required to address the following points:

- Difference between stop and wait, go back N and selective repeat.
- Comparison of the disadvantages of the different ARQ mechanisms.

MARKS DISTRIBUTION

Component	Maximum Marks	Marks Obtained
Preparation of Document	7	
Results	7	
Viva	6	
Total	**20**	

Submitted by: SATYAJIT GHANA

Register No: 17ETCS002159

25. Algorithm/Flowchart

Go Back N ARQ-

- Enter the size of the frames i.e. the sequence number.
- The transmitted frames will be equal to the $2^f - 1$ where f is the sequence number.
- The data to be transmitted can be generated using a loop and stored in the array arr[n].
- The timer for the acknowledgement can be set by using the data type clock in the library <time.h>.
- In case of Go back N ARQ the data is sent in a set of frames which is equal to transmitted frame length discussed earlier. Thus, a for loop is used to sent the first window of frames.
- The acknowledgement can generate any number between the window length which will denote the number of frames received correctly.
- The received frames will be given by using a for loop from the previous received (received initially at 0) to the received plus acknowledged frame thus if the frame id not acknowledged it is sent again.
- If the time taken to receive acknowledgement is less than the timer then the frame is sent again.
- This is repeated for all the frames using a while loop.

Selective Repeat ARQ-

- Enter the size of the frames i.e. the sequence number.
- The transmitted frames will be equal to the $2^f - 1$ where f is the sequence number.
- The data to be transmitted can be generated using a loop and stored in the array arr[n].
- The timer for the acknowledgement can be set by using the data type clock in the library <time.h>.
- In case of selective repeat the frame for whose negative acknowledgement NAK is received is sent again.
- To avoid sorting we are sending the frames one at a time.
- The rest is same as Go Back N.

26. Program

main.cpp

```
#include <iostream>
#include <vector>

template<typename T>
std::ostream& operator<<(std::ostream& out, std::vector<T> vec) {
    out << "[ ";
    for (auto& e : vec) {
        out << e << ", ";
    }
    out << "]" << std::endl;

    return out;
}

int main(int, char**) {
    using namespace std;
    cout << "Go back N " << endl;
    int window_size, tot_elems;
    cout << "Enter the Window size and Number of Elements : ";
    cin >> window_size >> tot_elems;
    // cout << "Enter the " << tot_elems << " Elements : ";
    std::vector<int> message(tot_elems);
    for (int i = 0 ; i < message.size() ; i++) {
        message[i] = i;
    }

    int N = window_size;
    std::vector<int>::iterator start = message.begin();
    std::vector<int>::iterator end = message.begin() + N >= message.end() ? message.end()
 : message.begin() + N;

    while (end <= message.end()) {
        std::vector<int> to_trasmit(start, end);

        cout << "TRANSMITTING " << to_trasmit;
        // for (auto& e : to_trasmit) {
        //     cout << e << " ";
        // }
        // cout << "]" << endl;

        int error_at;
        cout << "Enter the frame index if there was a bit error elese enter -1 : ";
```


```
        cin >> error_at;

        auto old_start = start;
        // error occured
        if (error_at != -1) {
            start = message.begin() + error_at;
        } else { // no error has occured
            start = end;
        }

        std::vector<int> recieved(old_start, start);
        cout << "RECIEVED : " << recieved;

        // check if we successfully transmitted the last bit
        if (end == message.end())
            break;

        end = start + N >= message.end() ? message.end() : start + N;
    }
}
```

27. Results

```
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab05
 /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab05/build/Lab05
Go back N
Enter the Window size and Number of Elements : 5 20
TRANSMITTING [ 0, 1, 2, 3, 4, ]
Enter the frame index if there was a bit error elese enter -1 : -1
RECIEVED : [ 0, 1, 2, 3, 4, ]
TRANSMITTING [ 5, 6, 7, 8, 9, ]
Enter the frame index if there was a bit error elese enter -1 : 7
RECIEVED : [ 5, 6, ]
TRANSMITTING [ 7, 8, 9, 10, 11, ]
Enter the frame index if there was a bit error elese enter -1 : 8
RECIEVED : [ 7, ]
TRANSMITTING [ 8, 9, 10, 11, 12, ]
Enter the frame index if there was a bit error elese enter -1 : -1
RECIEVED : [ 8, 9, 10, 11, 12, ]
TRANSMITTING [ 13, 14, 15, 16, 17, ]
Enter the frame index if there was a bit error elese enter -1 : -1
RECIEVED : [ 13, 14, 15, 16, 17, ]
TRANSMITTING [ 18, 19, ]
Enter the frame index if there was a bit error elese enter -1 : -1
RECIEVED : [ 18, 19, ]
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab05
```

Figure 0-1 OUTPUT for Go-Back-N

28. Analysis and Discussions

The main purpose of this experiment was to execute the receiver end program for the different ARQ protocols there are basically three types of protocol out of which two are using the sliding window protocol and the other one uses the static protocol.

In go back and N protocol the receiver has a window size of one and the sender has a window size of 2^m-1. The sender sends all the frames in that window within the given time period one after the other now the sender will wait for an acknowledgment from the receiver. The receiver can send a cumulative acknowledgment saying that all the frames till that region was received. Now the sender and receiver window will be updated depending on the acknowledgement receiver in the event that a frame or acknowledgment is lost and no acknowledgement is received by the received. Then the sender will retransmit all the frames in that window. This is better than go back and n looking at the fact that number of retransmissions is less and also the rate of transmission is higher. But there is a disadvantage that there is still a really high chance of duplication in a noisy channel.

In case of selective repeat, the sender and the received has the same window size which is defined by using the formula $2^m/2$. The sender sends all the data in the given window one after the other. Now the received can store all these data received in the given position in the window so even if the data comes out of order it is not a problem. Acknowledgement all the data which is addressed is sent by the receiver and in the event a frame is lost and the next frame arrives then the receiver will send a negative acknowledgment of the sender asking for only the frame which was not receiver by the receiver. In this way both bandwidth and the probability of duplication of data can be overcome. This is an effective method of transmission in a noisy channel along with packet switching.

- **Difference between stop and wait, go back N and selective repeat:**

STOP AND WAIT	GO BACK N	SELECTIVE REPEAT
Stop-and-wait ARQ, also referred to as alternating bit protocol, is a method in telecommunications to send information between two connected devices. It ensures that information is not lost due to dropped packets and that packets are received in the correct order. It is the simplest automatic repeat-request (ARQ) mechanism. A stop-and-wait ARQ sender sends one frame at a time.	"Go-Back-N Protocol is one of the sliding window protocols. The sliding window protocol is primarily an error control protocol, i.e. it is a method of error detection and error correction. The "go-back-n protocol" retransmits all the frames that lie after the frame which is damaged or lost.	"Selective Repeat Protocol" is one of the sliding window protocols. It retransmits only those frames that are suspected to lost or damaged.
In this case the window size at the transmitter and the receiver is equal to one or greater than one respectively.	Here the window size is N-1.	Here the window size is <= (N+1)/2
It only sends acknowledgment for the packet received by the receiver. It doesn't send negative acknowledgment.	It doesn't send negative acknowledgment for the data that is not received or the for the data that have any error. It only sends acknowledgment.	It sends a negative acknowledgment for the lost frames.
It has more delay time and is inefficient as compared to the other two ARQ mechanisms.	It requires large bandwidth.	It requires buffer space.

- **Comparison of the disadvantages of the different ARQ mechanisms:**

STOP AND WAIT	GO BACK N	SELECTIVE REPEAT
Window size at sender and receiver is equal to 1. (disadvantage)	Sender window size is 2^n-1 and the receiver window size is 1.	Sender window and the receiver window size is equal to 2^n-1.
There is no pipelining. (disadvantage)	Pipelining is implemented.	Pipelining is implemented.
It takes more delay time i.e. has large delay time. (disadvantage)	It requires large bandwidth. (disadvantage)	It requires buffer space. (disadvantage)
Sorting is not required, the data is received in the correct order.	Sorting is not required, the data is received in the correct order.	Receiver may receive the packets out of sequence which is one of the disadvantages and therefore sorting is required.

29. Conclusions

In this lab experiment we discussed the concept of ARQ. We conclude that ARQ mechanism is an error-control method for data transmission that uses acknowledgements and timeouts to achieve reliable data transmission over an unreliable service. The different kinds of ARQ mechanism as mentioned above are stop and wait, go back N and selective repeat and all these three are a kind of sliding window protocol. It can be concluded that out of all the three protocols the selective repeat protocol is the most efficient because it sends a negative acknowledgment and in addition to this it sends only the frame that has been not received by the receiver (i.e. sends only that frame that is suspected to be lost or damaged and therefore avoids the delay time but it needs buffer space. On the other hand, the stop and wait protocol has more delay time and is not efficient if the frame is thick and long and it does not send a negative acknowledgment for the lost or damaged frames. Similarly, the go back N protocol also doesn't send negative acknowledgment and in addition to this it retransmits all the frames that lie after the frame which is damaged or lost and thus wastes bandwidth.

30. Comments

a. Limitations of the experiment

The experiment does not specify how the frames might be dropped while they are being transmitted, here we've taken user input, instead they can be dropped randomly on a poison distribution.

b. Limitations of the results obtained

In this lab we learnt about the concept of different ARQ mechanism and also about its implementation. The design issue of data link layer one among is flow control. The disadvantages and the methods to overcome these protocols. About stop and wait with its disadvantages

c. Learning

We learnt the different ARQ Mechanisms, along with their advantages and disadvantages.

d. Recommendations

To simulate the packet drops, we an use poison distribution, this will give a proper network simulation result to this experiment.

Experiment 6: Socket Programming-I

Aim: To use TCP Sockets for Inter Process Communication

Objective: After carrying out this experiment, students will be able to:

- Apply TCP Socket programming technique to establish IPC between remote processes
- Analyse the difference between sockets and other enabling techniques for IPC such as Pipes and Message Queues

Problem statement: You are required to write programs to implement a TCP based echo server. The functionality of this server is that it should echo any data it receives from a client back to it.

Analysis: While analyzing your program, you are required to address the following points:

- How is socket programming different from other techniques for IPC such as Pipes and Message Queues?
- What happens if the number of incoming client requests exceeds the second argument of the `listen()` function in the server?

MARKS DISTRIBUTION

Component	Maximum Marks	Marks Obtained
Preparation of Document	7	
Results	7	
Viva	6	
Total	**20**	

Submitted by: SATYAJIT GHANA

Register No: 17ETCS002159

31. Algorithm/Flowchart

Client code description:

- Include all the standard libraries and the libraries required for socket programming like <netb.h> which is responsible for reserving highest port internet number, <sys/socket.h>,
- Create a structure of socket

struct sockaddr_in server

listen_fd = socket(AF_INET, SOCK_STREAM, 0)

bzero(&servaddr, sizeof(servaddr))

servaddr.sin_family = AF_INET

servaddr.sin_addr.s_addr = htons(INADDR_ANY)

servaddr.sin_port = htons(8080)

bind(listen_fd, (struct sockaddr *) &servaddr, sizeof(servaddr))

listen(listen_fd, 10)

- Echo the message back to the server using commands like accept(), with appropriate arguments.

Server code description:

- Include all the standard and necessary files required for socket programming.
- Create a structure for the server as well by using struct data-type.
- Pass the main string that needs to be echoed back by the client.

while(1)

bzero(sendline,100)

bzero(recvline,100)

fgets(sendline,100,stdin)

write(sockfd,sendline,strlen(sendline)+1)

read(sockfd,recvline,100)

printf("From Server: %s",recvline)

32. Program

```
// Client side C/C++ program to demonstrate Socket programming
#include <iostream>
#include <string>

#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

#define PORT 8080
#define IPADDR "127.0.0.1"
#define BUFSIZE 1024

/**
 * Client Side code, connects to the server,
 * sends the input from stdin to server
 * recieves from server and prints the data
*/
int main(int argc, char const *argv[]) {
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char buffer[BUFSIZE] = {0};

    std::cout << "CLIENT" << std::endl;

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

    // create the structure member values
    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, IPADDR, &serv_addr.sin_addr) <= 0) {
```


```
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }

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

    // infinite loop
    while (true) {
        std::string hello;

        std::cout << "Enter the Message to be sent to server : ";
        std::cin >> hello;

        send(sock, hello.c_str(), hello.size(), 0);

        std::cout << "Message sent to Server !\n"
                  << std::endl;

        valread = read(sock, buffer, BUFSIZE);
        printf("Recieved from Server : %s\n\n", buffer);

        // zero the buffer
        bzero(buffer, sizeof(buffer));
    }

    return 0;
}

#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

#include <iostream>
#include <string>

#define PORT 8080
#define BUFSIZE 1024
```


```
/**
 * Server Side code, takes input from stdin and sends it to the client
 * Recieves data from client and prints it.
*/
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[BUFSIZE] = {0};

    std::cout << "SERVER" << std::endl;

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

    // Forcefully attaching socket to the port 8080
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                   &opt, sizeof(opt))) {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }

    // assign the structure members
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Forcefully attaching socket to the port 8080
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    // listen to connections, max 3
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    // accept connection
```


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

    // keep running until the server or the client is closed
    while (true) {
        valread = read(new_socket, buffer, BUFSIZE);
        printf("Recieved from Client : %s\n\n", buffer);

        // zero the buffer
        bzero(buffer, sizeof(buffer));

        std::string hello;

        std::cout << "Enter the message to be sent to client : ";
        std::cin >> hello;

        send(new_socket, hello.c_str(), hello.size(), 0);
        printf("Message sent to Client !\n\n");
    }

    return 0;
}
```

33. Results

```
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-W  SIGINT(2)  79%  0.00K RAM  20:39:30
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab06/Lab06Server/build/Lab06Server
SERVER
Recieved from Client : Hi

Enter the message to be sent to client : meow
Message sent to Client !

Recieved from Client : boww

Enter the message to be sent to client : 1
Message sent to Client !

Recieved from Client : 2

Enter the message to be sent to client : ^C
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-W  SIGINT(2)  79%  0.00K RAM  20:39:52
```

Figure 0-1 Server Output

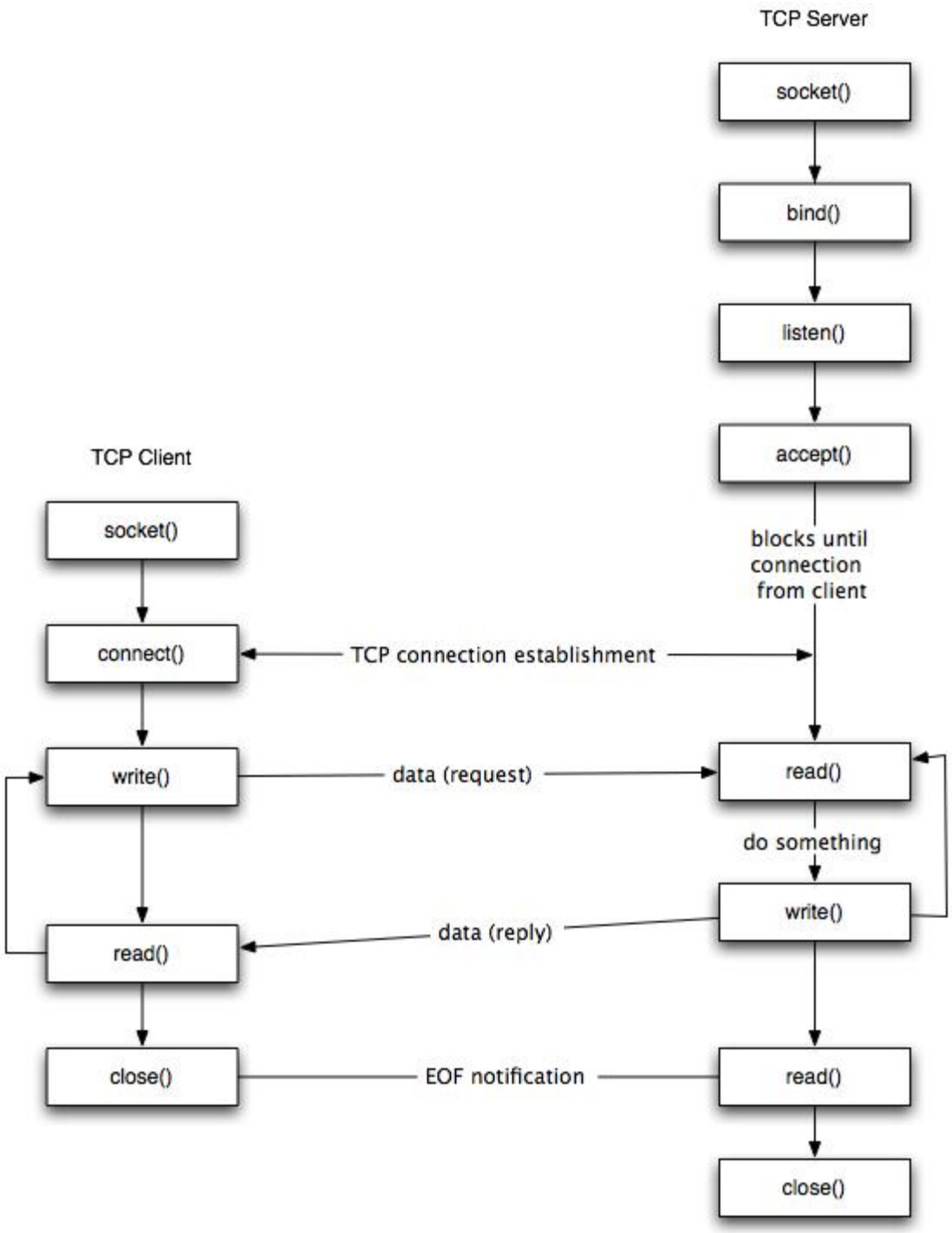
Figure 0-2 Client Output

34. Analysis and Discussions

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

Pipes are used as a medium to simply exchange information between sender and receiver. The messages that are exchanged are not analyzed or processed. Pipes are used in inter-process communication.

Message queues – In this, a queue is used to store the messages which are being exchanged between the sender and the receiver. A process inputs a message in the queue which allows another process to read it. An interface is provided to the processes and these processes use this interface to access the message queue to either put one message in the queue for other multiple processes to read it or read one message from the queue.



- **Socket creation:**

```
int sockfd = socket(domain, type, protocol)
```

sockfd: socket descriptor, an integer (like a file-handle)

domain: integer, communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)

type: communication type

SOCK_STREAM: TCP(reliable, connection oriented)

SOCK_DGRAM: UDP(unreliable, connectionless)

protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

- **Setsockopt:**

```
int setsockopt(int sockfd, int level, int optname,  const void *optval, socklen_t optlen);
```

This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.

- **Bind:**

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.

- **Listen:**

```
int listen(int sockfd, int backlog);
```

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending

connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

- **Accept:**

```
int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data.

Stages for Client

- **Socket connection:** Exactly same as that of server's socket creation
- **Connect:**
- int connect(int sockfd, const struct sockaddr *addr,

  socklen_t addrlen);

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

35. Conclusions

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else.

To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes.

Sockets were first introduced in 2.1BSD and subsequently refined into their current form with 4.2BSD. The sockets feature is now available with most current UNIX system releases.

36. Comments

a. Limitations of the experiment

The experiment deals with only transmitting strings over the network, since they are simpler to transmit, transmitting different data structures and retaining the type of data is difficult, but justifies the complex real-world network transfer.

b. Limitations of the results obtained

The connection here is limited to only one, to test the experiment properly, a few more connections can be made.

c. Learning

We learnt how to setup simple socket connection over the network and transmit strings.

d. Recommendations

Something like Google's protobuf can be used to transmit different data structures

Experiment 7: DLL ARQ Mechanisms using TCP Sockets

Aim: To use TCP Sockets to implement the ARQ mechanisms at the Data Link Layer

Objective: After carrying out this experiment, students will be able to:

- Apply TCP Socket programming technique to implement the ARQ mechanisms at the Data Link Layer

Problem statement: You are required to write programs to implement a TCP based server that receives frames sent to it by a client. The functionality of this server is that it should echo any data it receives from a client back to it. You are required to implement stop and wait, go back N and selective repeat ARQ mechanisms. Consider that you have to transmit and receive a total of 20 frames using $W_T=W_R=1$, $W_T=5$ and $W_R=1$ and $W_T=W_R=5$ for stop and wait, go back N and selective repeat respectively.

Analysis: While analyzing your program, you are required to address the following points:

- How does the functionality of the program differ when you have the `accept()` function call at the server within an infinite loop as opposed to having it outside?

MARKS DISTRIBUTION

Component	Maximum Marks	Marks Obtained
Preparation of Document	7	
Results	7	
Viva	6	
Total	**20**	

Submitted by:

Register No:

37. Algorithm/Flowchart

Go-Back-N:

```
N  =
window
size
          Sn = sequence number
          Sb = sequence base
          Sm = sequence max
          ack = ack number
          nack = first non acknowledged
          Receiver:
          Do the following forever:
            Randomly accept or reject packet

            If the packet received and the packet is error free
                  Accept packet
                  Send a positive ack for packet
            Else
                  Refuse packet
                  Send a negative ack for packet

          Sender:
          Sb = 0
          Sm = N – 1
          ack = 0
          Repeat the following steps forever:
            Send packet with ack
            If positively ack is recieved:
                   ack++
                   Transmit a packet where Sb <= ack <= Sm.
                   packets are transmitted in order
            Else
                    Enqueue the nack into the queue
            //check if last packet in the window is sent
            if(ack==Sm)
              if(queue is not empty)
                   // start from the first nack packet
                  nack = queue.front();
                  empty the queue
                  ack = nack
```


```
    Sm = Sm + (ack – Sb)
    Sb = ack
```

Selective Repeat:

```
N  =
window
size
        Sn = sequence number
        Sb = sequence base
        Sm = sequence max
        ack = ack number
        nack = first non acknowledged
        Receiver:
        Do the following forever:
          Randomly accept or reject packet

          If the packet received and the packet is error free
                Accept packet
                Send a positive ack for packet
          Else
                Refuse packet
                Send a negative ack for packet

        Sender:
        Sb = 0
        Sm = N – 1
        ack = 0
        Repeat the following steps forever:
          If the packet was not already positively acknowledged by receiver
                  Send packet with ack
                  If positively ack is recieved:
                       Transmit a packet where Sb <= ack <= Sm.
                       packets are transmitted in order
                  Else
                       Enqueue the nack into the queue
                  ack++
          //check if last packet in the window is sent
          if(ack==Sm)
            if(queue is not empty)
```


```
                // start from the first nack packet
                nack = queue.front();
                empty the queue
                ack = nack

            Sm = Sm + (ack - Sb)
            Sb = ack
```

38. Program

Client:

```
// Client side C/C++ program to demonstrate Socket programming
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 8080

#include <iostream>

int main(int argc, char const *argv[]) {
    std::cout << "Enter the ARQ Mechanism to use " << std::endl;
    std::cout
        << "1.Stop And Wait\t2.Go Back N\3.Selective Repeat\nYour Choice : "
        << std::endl;
    int choice;
    std::cin >> choice;
    std::cin.ignore();

    int sockfd = 0, valread;
    struct sockaddr_in serv_addr;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        exit(EXIT_FAILURE);
    } else {
        std::cout << argv[0] << " : SOCKET CREATED " << std::endl;
    }

    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;
} else {
    std::cout << argv[0] << " : ADDRESS CONVERTED" << std::endl;
}

if (connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
    printf("\nConnection Failed \n");
    return -1;
} else {
    std::cout << argv[0] << " : CONNECTION ESTABLISHED" << std::endl;
}

char buffer[1024] = {0};

// create the data (20frames)
std::string frames("");

for (int i = 0; i < 20; i++) frames.push_back(std::rand() % 10 + '0');

std::cout << argv[0] << " : FRAMES TO TRANSMIT\n" << frames << std::endl;

switch (choice) {
    case 1: {
    } break;
    case 2: {
        int windowSize;
        std::cout << "Enter the window size : ";
        std::cin >> windowSize;
        std::cin.ignore();
        int N = windowSize;

        std::string::iterator start = frames.begin();
        std::string::iterator end = frames.begin() + N >= frames.end()
                                        ? frames.end()
                                        : frames.begin() + N;

        while (true) {
            std::string to_transmit(start, end);
            std::cout << "TRANSMITTING : " << to_transmit << std::endl;

            if (to_transmit.empty()) {
```


```
                    break;
                }

                write(sockfd, to_transmit.c_str(), to_transmit.size());

                bzero(buffer, sizeof(buffer));
                read(sockfd, buffer, sizeof(buffer));
                std::string recieved(buffer);

                std::cout << "SERVER RECIEVED : " << recieved << std::endl;

                // error has occured
                int idx = recieved.size();

                start += idx;
                end = start + N >= frames.end() ? frames.end() : start + N;
            }

        } break;
        case 3: {
        } break;
        default: {
            send(sockfd, "EXIT", sizeof("EXIT"), 0);
            close(sockfd);
            exit(EXIT_SUCCESS);
        }
    }

    close(sockfd);
    exit(EXIT_SUCCESS);

    return 0;
}
```

Server:

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


```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <vector>

int main(int argc, char const *argv[]) {
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    } else {
        std::cout << argv[0] << " : "
                  << "SOCKET CREATED SUCCESSFULLY" << std::endl;
    }

    // Forcefully attaching socket to the port 8080
    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);

    // Forcefully attaching socket to the port 8080
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    } else {
        std::cout << argv[0] << " : "
                  << "BIND SUCCESSFUL TO PORT " << PORT << std::endl;
    }

    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    } else {
        std::cout << argv[0] << " : "
```


```
                << "NOW LISTENING" << std::endl;
    }

    if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                             (socklen_t *)&addrlen)) < 0) {
        perror("accept");
        exit(EXIT_FAILURE);
    } else {
        std::cout << argv[0] << " : CLIENT ACCEPTED" << std::endl;
    }

    char buffer[1024] = {0};

    // fetch the operation from the client
    while (true) {
        bzero(buffer, 1024);

        valread = read(new_socket, buffer, sizeof(buffer));
        std::string operation(buffer);
        if (operation.empty()) {
            break;
        }

        std::cout << argv[0] << " : RECIEVED : " << buffer << std::endl;

        std::cout << "Enter the index of the byte from above frames which had "
                     "error\nenter -1 if there was no error :  ";
        int idx;
        std::cin >> idx;
        std::cin.ignore();

        std::string recieved(buffer);
        if (idx != -1) {
            recieved = recieved.substr(0, idx);
        }

        write(new_socket, recieved.c_str(), recieved.size());
    }

    // Close the Server File Descriptor
    close(server_fd);

    return 0;
}
```


39. Results

```
shadowleaf@SHADOWLEAF-ROG > /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Client
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Client/build/Lab06Client
Enter the ARQ Mechanism to use
1.Stop And Wait 2.Go Back N.Selective Repeat
Your Choice :
2
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Client/build/Lab06Client : SOCKET CREATED
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Client/build/Lab06Client : ADDRESS CONVERTED
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Client/build/Lab06Client : CONNECTION ESTABLISHED
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Client/build/Lab06Client : FRAMES TO TRANSMIT
36753562912709360626
Enter the window size : 5
TRANSMITTING : 36753
SERVER RECIEVED : 36753
TRANSMITTING : 56291
SERVER RECIEVED : 56291
TRANSMITTING : 27093
SERVER RECIEVED : 27093
TRANSMITTING : 60626
SERVER RECIEVED : 60626
shadowleaf@SHADOWLEAF-ROG > /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Client
```

```
shadowleaf@SHADOWLEAF-ROG > /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Server
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Server/build/Lab06Server
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Server/build/Lab06Server : SOCKET CREATED SUCCESSFULLY
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Server/build/Lab06Server : BIND SUCCESSFUL TO PORT 8080
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Server/build/Lab06Server : NOW LISTENING
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Server/build/Lab06Server : CLIENT ACCEPTED
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Server/build/Lab06Server : RECIEVED : 36753
Enter the index of the byte from above frames which had error
enter -1 if there was no error :  -1
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Server/build/Lab06Server : RECIEVED : 56291
Enter the index of the byte from above frames which had error
enter -1 if there was no error :  -1
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Server/build/Lab06Server : RECIEVED : 27093
Enter the index of the byte from above frames which had error
enter -1 if there was no error :  -1
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Server/build/Lab06Server : RECIEVED : 60626
Enter the index of the byte from above frames which had error
shadowleaf@SHADOWLEAF-ROG > /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab07/Lab07Server
```

40. Analysis and Discussions

accept - accept a new connection on a socket

```
#include <sys/socket.h>

int accept(int socket, struct sockaddr *restrict address,
       socklen_t *restrict address_len);
```

The *accept*() function shall extract the first connection on the queue of pending connections, create a new socket with the same socket type protocol and address family as the specified socket, and allocate a new file descriptor for that socket.

Calling the accept function in an infinite loop will keep creating sockets for every new connection if possible, otherwise it will fail.

41. Conclusions

Stop and Wait –

The sender sends the packet and waits for the ACK (acknowledgement) of the packet. Once the ACK reaches the sender, it transmits the next packet in row. If the ACK is not received, it re-transmits the previous packet again.

Go Back N –

The sender sends N packets which is equal to the window size. Once the entire window is sent, the sender then waits for a cumulative ACK to send more packets. On the receiver end, it receives only in-order packets and discards out-of-order packets. As in case of packet loss, the entire window would be re-transmitted.

Selective Repeat –

The sender sends packet of window size N and the receiver acknowledges all packet whether they were received in order or not. In this case, the receiver maintains a buffer to contain out-of-order packets and sorts them. The sender selectively re-transmits the lost packet and moves the window forward.

PROPERTIES	STOP AND WAIT	GO BACK N	SELECTIVE REPEAT
Sender window size	1	N	N
Receiver Window size	1	1	N
Minimum Sequence number	2	N+1	2N
Efficiency	1/(1+2*a)	N/(1+2*a)	N/(1+2*a)
Type of Acknowledgement	Individual	Cumulative	Individual

PROPERTIES	STOP AND WAIT	GO BACK N	SELECTIVE REPEAT
Supported order at Receiving end	–	In-order delivery only	Out-of-order delivery as well
Number of retransmissions in case of packet drop	1	N	1

42. Comments

a. Limitations of the experiment

The experiment is limited to 20 packets, which is very learn to simulate an ARQ mechanism for comparison.

b. Limitations of the results obtained

The results are obtained for very low number of packets, hence the packet dropping cannot be simulated well enough to learn the advantages and disadvantages of the individual algorithms.

c. Learning

We learnt how to implement sliding window ARQ mechanism using TCP Socket Programming.

d. Recommendations

Instead of reading the input from STDIN, introduce random packet drops from a poisson distribution to simulate a proper network, which will show the proper advantages of the different algorithms under different conditions.

Experiment 8: Alphanumeric Cipher using TCP Sockets

Aim: To use TCP Sockets to implement simple alphanumeric ciphers

Objective: After carrying out this experiment, students will be able to:

- Apply simple alphanumeric cipher techniques to encrypt and decrypt text messages

Problem statement: You are required to write programs for a TCP client-server system that performs simple encryption and decryption using alphanumeric ciphers. The client should first authenticate a user using a pre-defined password. Then the client (user) sends a message to the server. The server will encrypt the message using alphanumeric cipher and store it in a file. Subsequently, when a user asks for the message he has entered previously, the message should be decrypted and transmitted back to the client.

Analysis: While analyzing your program, you are required to address the following points:

- From the security perspective, how can this implementation be made more secure?
- What are the vulnerabilities of alphanumeric ciphers?

MARKS DISTRIBUTION

Component	Maximum Marks	Marks Obtained
Preparation of Document	7	
Results	7	
Viva	6	
Total	**20**	

Submitted by:

Register No:

43. Algorithm/Flowchart

44. Program

CLIENT CODE:

```
// Client side C/C++ program to demonstrate Socket programming
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#define PORT 8080

#include <iostream>

bool authenticated() {
    using namespace std;

    std::string pass("meow123");

    cout << "Enter the auth pass : ";
    std::string input;
    cin >> input;

    return pass == input;

}

int main(int argc, char const *argv[]) {
    if (not authenticated()) {
        std::cout << "AUTH ERROR !" << std::endl;
        exit(EXIT_FAILURE);
    }

    int sockfd = 0, valread;
    struct sockaddr_in serv_addr;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        exit(EXIT_FAILURE);
    } else {
        std::cout << argv[0] << " : SOCKET CREATED " << std::endl;
    }
```


```
    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;
    } else {
        std::cout << argv[0] << " : ADDRESS CONVERTED" << std::endl;
    }

    if (connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        printf("\nConnection Failed \n");
        return -1;
    } else {
        std::cout << argv[0] << " : CONNECTION ESTABLISHED" << std::endl;
    }

    char buffer[1024] = {0};

    while(true) {

        std::cout << "1.STORE\t2.FETCH\t3.EXIT\nCHOICE : ";
        int choice;
        std::cin >> choice;

        switch(choice) {
            case 1: {
                std::cout << "Enter the message to send : " << std::endl;
                std::string input;
                std::cin.ignore();
                std::getline(std::cin, input);

                send(sockfd, "STO", sizeof("STO"), 0);
                send(sockfd, input.c_str(), input.size(), 0);
                std::cout << argv[0] << " : INPUT MESSAGE SENT !" << std::endl;

                bzero(buffer, sizeof(buffer));
                recv(sockfd, buffer, sizeof(buffer), 0);

                std::cout << argv[0] << " : RECIEVED FROM SERVER : " << buffer << std::en
dl;
            } break;
            case 2: {
```


```
                send(sockfd, "FET", sizeof("FET"), 0);
                std::cout << argv[0] << " : FETCH INSTRUCTION SENT !" << std::endl;

                bzero(buffer, sizeof(buffer));
                recv(sockfd, buffer, sizeof(buffer), 0);

                std::cout << argv[0] << " : RECIEVED FROM SERVER : " << buffer << std::en
dl;
            } break;
            default: {
                send(sockfd, "EXIT", sizeof("EXIT"), 0);
                close(sockfd);
                exit(EXIT_SUCCESS);
            }
        }
    }

    return 0;
}
```

SERVER CODE:

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

#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>

unsigned long long pow36[11];

void calc_pow_36();
unsigned long long base36encode(std::string);
std::string base36decode(unsigned long long message);

char base36chars[37] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int main(int argc, char const *argv[]) {
    calc_pow_36();
```


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

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    } else {
        std::cout << argv[0] << " : " << "SOCKET CREATED SUCCESSFULLY" << std::endl;
    }

    // Forcefully attaching socket to the port 8080
    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 );

    // Forcefully attaching socket to the port 8080
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }  else {
        std::cout << argv[0] << " : " << "BIND SUCCESSFUL TO PORT " << PORT << std::endl;
    }

    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    } else {
        std::cout << argv[0] << " : " << "NOW LISTENING" << std::endl;
    }

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


```
        std::cout << argv[0] << " : CLIENT ACCEPTED" << std::endl;
    }

    char buffer[1024] = {0};

    // fetch the operation from the client
    while (true) {
        bzero(buffer, 1024);

        valread = read( new_socket , buffer, sizeof(buffer));
        std::string operation(buffer);

        std::cout << argv[0] << " : OPERATION RECIEVED : " << buffer << std::endl;

        if (operation == "STO") {
            std::ofstream outfile;
            outfile.open("STORE.dat", std::ios::out);

            bzero(buffer, 1024);
            read(new_socket, buffer, sizeof(buffer));
            std::string recieved(buffer);

            unsigned long long encoded = base36encode(recieved);

            if (encoded == 0) {
                write(new_socket, "FAILED", sizeof("FAILED"));
            } else {
                write(new_socket, "SUCCESS", sizeof("SUCCESS"));
                outfile << encoded;
                std::cout << argv[0] << " : ENCODED : " << encoded << std::endl;
            }
            outfile.close();
        } else if (operation == "FET") {
            std::ifstream infile;
            infile.open("STORE.dat", std::ios::in);
            std::string val;
            infile >> val;

            std::string decoded_val = base36decode(std::stoull(val));
            write(new_socket, decoded_val.c_str(), decoded_val.size());

            infile.close();
        } else {
            break;
        }
```


```
    }

    // Close the Server File Descriptor
    close(server_fd);

    return 0;
}

void calc_pow_36() {
    pow36[0] = 1;

    for (int i = 0 ; i <= 9 ; i++) {
        pow36[i+1] = pow36[i] * 36ull;
    }
}

unsigned long long base36encode(std::string message) {
    unsigned long long n;
    unsigned long long encoded = 0;

    std::reverse(message.begin(), message.end());

    int i = 0;
    for (char& c : message) {
        if (c >= '0' and c <= '9') {
            n = c - '0';
        } else if (c >= 'A' and c <= 'Z') {
            n = c - 'A' + 10;
        } else if (c >= 'a' and c <= 'z') {
            n = c - 'a' + 10;
        } else {
            return 0;
        }
        encoded += pow36[i] * n;
        i++;
    }

    return encoded;
}

std::string base36decode(unsigned long long message) {
    std::string decoded("");

    while(message != 0) {
        decoded.push_back(base36chars[message % 36]);
```


```
        message /= 36;
    }

    std::reverse(decoded.begin(), decoded.end());

    return decoded;
}
```

45. Results

```
shadowleaf@SHADOWLEAF-ROG /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Server
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Server/build/Lab06Server
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Server/build/Lab06Server : SOCKET CREATED SUCCESSFULLY
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Server/build/Lab06Server : BIND SUCCESSFUL TO PORT 8080
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Server/build/Lab06Server : NOW LISTENING
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Server/build/Lab06Server : CLIENT ACCEPTED
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Server/build/Lab06Server : OPERATION RECIEVED : STO
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Server/build/Lab06Server : ENCODED : 1045472
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Server/build/Lab06Server : OPERATION RECIEVED : FET
shadowleaf@SHADOWLEAF-ROG /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Server
```

```
shadowleaf@SHADOWLEAF-ROG /mnt/d/University-Work/University-Work-SEM-05 79% 0.00K RAM 21:02:58
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Client/build/Lab06Client
Enter the auth pass : meow123
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Client/build/Lab06Client : SOCKET CREATED
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Client/build/Lab06Client : ADDRESS CONVERTED
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Client/build/Lab06Client : CONNECTION ESTABLISHED
1.STORE 2.FETCH 3.EXIT
CHOICE : 1
Enter the message to send :
meow
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Client/build/Lab06Client : INPUT MESSAGE SENT !
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Client/build/Lab06Client : RECIEVED FROM SERVER : SUCCESS
1.STORE 2.FETCH 3.EXIT
CHOICE : 2
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Client/build/Lab06Client : FETCH INSTRUCTION SENT !
/mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Client/build/Lab06Client : RECIEVED FROM SERVER : MEOW
1.STORE 2.FETCH 3.EXIT
CHOICE : 3
shadowleaf@SHADOWLEAF-ROG /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab08/Lab06Client
```

46. Analysis and Discussions

47. Conclusions

48. Comments

a. Limitations of the experiment
b. Limitations of the results obtained
c. Learning
d. Recommendations.