INDEX

NAME : A. Kavin STD : III nd year SEC. : CSE-B ROLL NO. 220701122

S.No.	Date	Title	Page No.	Teacher's Sign/ Remarks
1.	16/7/24	Study of various network commands used in Linux & windows		
2.	23/7/24	Study of Network cables		
3.	30/7/24	Experiments of CISCO PACKET TRACER (simulation tools		
4.	6/8/24	Setup and configure a LAN using a switch and Ethernet cable		
5.	9/8/24	Experiments on packet capture tool; wireshark		
6.	16/8/24	Error correction at data link layer (Hamming code)		
7.	23/8/24	Flow control at data link layer (Sliding window protocol		
8.	10/9/24	Stimulate virtual LAN		
		Cisco Packet Tracer		
9.	30/9/24	Implementation of subnetting in CISCO Packet tracer		
10.	4/10/24	Internetworking using router DHCP server and internet cloud		
11.	8/10/24	Stimulate static routing protocol		
		configuration using CISO Packet & RIP		
12.	15/10/24	Echo client TCP/UDP sockets chart client server TCP/UDP		
13	22/10/24	Write own Ping problem		
14.	25/10/24	Raw sockets to implement		
		Packet Sniffing		
15.	29/10/24	Webilizer tool		

Completed

Palaniappa

Exp. NO : 06 Practical-6

Date : 16/08/2024

Aim

Write a program to implement error detection and correction using Hamming code concept. Make a test run to input data stream and verify error correction features

Error correction at Data Link layer

Hamming code is a set of error correction code that can be used to detect and correct the errors that can occur when the data is transmitted from the sender to the receiver. It is a technique developed by R.W Hamming for error correction.

Create sender program with below features

1. Input to sender file should be text of any length. Program should convert the text to binary
2. Apply hamming code concept on the binary data and add redundant bits to it
3. Save this output in a file called channel.

Create a receiver program with below feat

1. Receiver program should read the input from channel file
2. Apply hamming code on binary data to check for errors
3. If there is an error, display the position of the error
4. Else remove the redundant bits and convert the binary data to ascii and display the output

Student Observation

code

```
import math
def char_to_binary(ch):
    binary = []
    for i in range(7, -1, -1):
        binary.append((ord(ch)>>i)&1)
    return binary
def calculate_parity_bits(hamming_code, n, r):
    for i in range(r):
        parity_pos = 2**i
        parity = 0
        for j in range(parity_pos, n+1, 2*parity_pos):
            for k in range(j, j+parity_pos):
                if k<=n:
                    parity^= hamming_code[k]
        hamming_code[parity_pos]= parity
def generate_hamming_code(data_bits, m):
    r=0
    n=m
    while n+r+1>2**r:
        r+=1
    n=m+r
    hamming_code =[0]*(n+1)
    j=0
    k=0
    for i in range(1, n+1):
        if i==2**k:
            k+=1
        else:
            hamming_code[i]= data_bits[j]
            j+=1
    calculate_parity_bits(hamming_code, n, r)
    return hamming_code, n, r
```

```
def detect_and_correct_error(hamming_code,n,r):
    error_pos=0
    for i in range(r):
        parity_pos=2**i
        parity=0
        for j in range(parity_pos, n+1, 2*parity_pos):
            for k in range(j, j+parity_pos):
                if k<=n:
                    parity^=hamming_code[k]
            if parity!=0:
                error_pos+=parity_pos
    return error_pos
def binary_to_char(binary):
    output=
    for i in range(0, len(binary), 8):
        ch=0
        for j in range(8)
            ch|=(binary[i+j]<<(7-j))
        output+=chr(ch)
    return output
def main():
    input_string=input("Enter the input string
    binary=[]
    for ch in input_string:
        binary.extend(char_to_binary(ch))
    data_bits=binary[:]
    hamming code, n, r=generate_hamming_code
                        (data_bits, len(data_bits))
    print("generated hamming code:", ' '.join
                (map(str, hamming_code[1:])))
```

```
error_pos = -1
while True:
    error_pos = int(input("Enter the position to
                stimulate error (0 for no error):"))
    if error_pos > 0 and any(error_pos == 2**k for
                k in range(r)):
        Print("Error cannot be introduced in a
                redundant (parity) position. Please
                choose another position.")
    elif error_pos < 0 or error_pos > n:
        Print("Invalid position. Please enter
                a position between 1 and ", n)
    else:
        break

if error_pos > 0:
    hamming_code[error_pos] ^= 1
    Print("Hamming code with error:", ''.join
        (map(str, hamming_code[1:])))
    detected_error_pos = detect_and_error_error
                (hamming_code, n, r)
    if detected_error_pos == 0:
        Print("No error detected.")
    else
        Print(f"Error detected at position:
                {detected_error_pos}")
        binary_error_pos = format(detected_
                error_pos, 'b')
        Print(f"corrected bit at position
{detected_error_pos} (binary: {binary_error_
    pos}): {hamming code[detected_error_pos]}")
    hamming_code[detected_error_pos] ^= 1
    Print("corrected hamming code:", ''.
        join(map(str, hamming code[1:])))
```

```
    corrected_data_bits = []
    k = 0
    for i in range(1, n+1):
        if i != 2**k:
            corrected_data_bits.append(hamming_code[i])
        else:
            k += 1
    corrected_string = binary_to_char(corrected_data_bits)
    print("corrected string:", corrected_string)

if __name__ == "__main__":
    main()
```

Output

```
Enter the input string : apple
Generated Hamming code: 1100110000010
1111000001111000001110110001100101
Enter the position to stimulate error : 3
Hamming code with error :1110110000010
1011110000011100000111000001110000
1110110001100101
Error detected at position : 3
corrected bit at position 3 (binary : 11) : 1
Corrected hamming code : 1100110000010
1111000001111000001110110001100101
corrected string : apple
```

Result

Thus the program is executed successfully and output is verified

16/8/24