16/08/2024

Experiment No: 6 Error correction at Data Link Layer

Aim:

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

Error correction at Data Link Layer:

Hamming code is a set of errors correction codes that can be used to detect and correct the errors that can occur when the data is transmitted from the senders 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 a text of any length. Program should convert the text to binary.
2. Apply hamming code concept on the binary data and add redundant bits on it.
3. Save this output in a file called channel

Create Receiver program with below features:

1. Receiver program should read the input from channel file.
2. Apply hamming code on the 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.

Program code:

```
def string_to_binary(input_string):
    return ''.join(format(ord(c), '08b') for c in input_string)

def binary_to_string(binary_data):
    chars = []
    for i in range(0, len(binary_data), 8):
        byte = binary_data[i:i+8]
        chars.append(chr(int(byte, 2)))
    return ''.join(chars)

def calculate_parity_bits(data):
    n = len(data)
    r = 0
    while (2**r) < (n+r+1):
        r += 1
    return r

def insert_parity_bits(data, r):
    n = len(data)
    j = 0
    k = 0
    m = n+r
    hamming_code = []
    for i in range(1, m+1):
        if i == 2**j:
            hamming_code.append('0')
            j += 1
        else:
            hamming_code.append(int(data[k]))
            k += 1
    return ''.join(map(str, hamming_code))
```

```
def calculate_parity_values(hamming_code, r):
    hamming_code = list(map(int, hamming_code))
    n = len(hamming_code)
    for i in range(r):
        parity_pos = 2**i
        parity_val = 0
        for j in range(1, n+1):
            if j & parity_pos and j != parity_pos:
                parity_val ^= hamming_code[j-1]
        hamming_code[parity_pos-1] = parity_val
    return ''.join(map(str, hamming_code))

def detect_and_correct_error(hamming_code, r):
    hamming_code = list(map(int, hamming_code))
    n = len(hamming_code)
    error_position = 0
    for i in range(r):
        parity_pos = 2**i
        parity_val = 0
        for j in range(1, n+1):
            if j & parity_pos:
                parity_val ^= hamming_code[j-1]
        if parity_val != 0:
            error_position += parity_pos
    if error_position:
        print(f"Error at: {error_position}")
        print(f"Binary pos: {bin(error_position)[2:].zfill(r)}")
        hamming_code[error_position-1] ^= 1
        print(f"Corrected code: {''.join(map(str, hamming_code))}")
```

```
        else:
            print("No error.")
    return ''.join(map(str, hamming_code))

def extract_data_from_hamming(hamming_code, r):
    j=0
    data=[]
    for i in range(1, len(hamming_code)+1):
        if i!=2**j:
            data.append(hamming_code[i-1])
        else:
            j+=1
    return ''.join(map(str, data))

def main:
    input_string = input("Enter the string:")
    binary_data = string_to_binary(input_string)
    print(f"Binary: {binary_data}")
    r = calculate_parity_bits(binary_data)
    hamming_code = insert_parity_bits(binary_data, r)
    hamming_code = calculate_parity_values
                   (hamming_code, r)
    print(f"Hamming code: {hamming_code}")
    redundant_bits = {2**i for i in range(r)}
    while True:
        print("\n Flip a bit for error...")
        error_bit = int(input(f" Flip bit
                    (1-{len(hamming_code)}): "))
        if error_bit in redundant_bits:
            print(" Redundant bit. Choose
                    another position")
```

```
        else:
            hamming_
            [i-1] if ham
            else '0') +
            print(f"Ham
            break
    hamming_code=

    corrected_bina

    corrected_strin
    print(f" Final
if __name__ =="__
    main()
```

Output:

```
Enter the stri
Binary: 01001
Hamming cod
Flip a bit fo
Flip bit (1-21)
Redundant bi
Flip a bit f
Flip bit (1-21
Error!
Hamming cod
Error at: 3
Binary pos: 0
corrected cod
Final outpu
Result:
```

10/9/24 Thus, for implement is successfull

```
        else:
            hamming_code = hamming_code[:error_bit-1] +
            ('1' if hamming_code[error_bit-1] == '0'
            else '0') + hamming_code[error_bit:]
        print(f"Hamming code with error:
                                {hamming_code}")
        break
    hamming_code = detect_and_correct_error
                    (hamming_code, r)
    corrected_binary_data = extract_data_from_hamming
                    (hamming_code, r)
    corrected_string = binary_to_string(corrected_binary_
                                        data)
    print(f"Final output : '{corrected_string}' ")
if __name__ == "__main__":
    main()
```

Output:

```
Enter the string: Hi
Binary: 0100100001101001
Hamming code: 0100100000110 0011 1001
Flip a bit for error....
Flip bit (1-21): 2
Redundant bit. Choose another position
Flip a bit for error....
Flip bit (1-21): 3
Error!
Hamming code with error: 0110100110000110010011
Error at: 3
Binary pos: 00011
corrected code: 0100100110000110010011
Final output: Hi
```

Result:

Thus, the program for Hamming code for implementing error detection and correction is successfully executed and output is verified.

10/9/24