Computer Netwroks Lab

Practical-3

Name: Vedant Bhutada

Roll: 69

Batch: A4

Aim: Data Link Layer Implementation

a) Implement Bit and Byte Stuffing method used by data link layer.

Also b) Write a program to Demonstrate CRC error detection technique using python/java programming

```
import random

def xor(a, b):
    result = []
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else:
            result.append('1')
    return ''.join(result)

def xordiv(dividend, divisor, sender=False):
    if sender:
        dividend += '0' * (len(divisor) - 1)

    pick = len(divisor)
    tmp = dividend[0: pick]

    while pick < len(dividend):
        if tmp[0] == '1':
            tmp = xor(divisor, tmp) + dividend[pick]
        else:
            tmp = xor('0' * pick, tmp) + dividend[pick]
        pick += 1

    if tmp[0] == '1':
        tmp = xor(divisor, tmp)
    else:
        tmp = xor('0' * pick, tmp)

    checkword = tmp
    return checkword

def encodeData(data, key):
    remainder = xordiv(data, key, True)
    return data + remainder

def print_changed_bit(original_data, received_data):
    for i in range(len(original_data)):
        if original_data[i] != received_data[i]:
            print(f"Bit {i + 1} changed: Original bit = {original_data[i]}, Received bit = {received_data[i]}")

def correct_data(original_data, received_data):
    corrected_data_list = list(received_data)
    for i in range(len(original_data)):
        if original_data[i] != received_data[i]:
            corrected_data_list[i] = original_data[i]
    corrected_data = ''.join(corrected_data_list)
    return corrected_data

def simulate_correct_received_data(data, key):
    encoded_data = encodeData(data, key)
    print("Encoded Data:", encoded_data)

    # No error introduced, data is valid
    rem = xordiv(encoded_data, key)
    if int(rem) == 0:
        print("Valid Data")
        #print("Received Data:", data)
    else:
        print("Invalid Data")

def simulate_error_received_data(data, key):
    encoded_data = encodeData(data, key)
    print("Encoded Data:", encoded_data)
    # Simulate an error by flipping a random bit
    error_position = random.randint(0, len(encoded_data) - 1)
    encoded_data_list = list(encoded_data)
    original_bit = encoded_data_list[error_position]
    encoded_data_list[error_position] = '1' if original_bit == '0' else '0'
    corrupted_data = ''.join(encoded_data_list)
    print("Received Data with Error : ", corrupted_data)

    # Print which bit is changed
    print_changed_bit(data, corrupted_data)

    # Correct the data
    corrected_data = correct_data(data, corrupted_data)
    print("Corrected Data:", corrected_data)

    # Check the received data for errors
    rem = xordiv(corrected_data, key)
    if int(rem) == 0:
        print("Data Corrected and Valid")
    else:
        print("Data Not Corrected and Invalid")

data = "1011001100"
key = "1101"

print("Sender Side----------------------------")
print("Data:", data)
print("Key:", key)
print("CRC:", xordiv(data, key, True))
print()

print("Receiver Side (Correct Data)-------------------------")
simulate_correct_received_data(data, key)

print("Receiver Side (Error Data)-------------------------")
simulate_error_received_data(data, key)
```

```
Sender Side----------------------------
Data: 1011001100
Key: 1101
CRC: 111

Receiver Side (Correct Data)-------------------------
Encoded Data: 1011001100111
Valid Data
Receiver Side (Error Data)-------------------------
Encoded Data: 1011001100111
Received Data with Error :  1011101100111
Bit 5 changed: Original bit = 0, Received bit = 1
Corrected Data: 1011001100111
Data Corrected and Valid
```

```
def polynomial_to_binary(poly_str):
    poly_str = poly_str.replace(" ", "")
    binary_str = ""

    coefficients = []
    terms = poly_str.split("+")

    for term in terms:
        if term == "1":
            degree = 0
        else:
            parts = term.split("x^")
            if len(parts) == 2:
                degree = int(parts[1])
            elif term == "x":
                degree = 1
            else:
                degree = 0

        if degree >= len(coefficients):
            coefficients.extend([0] * (degree - len(coefficients) + 1))

        coefficients[degree] = 1

    for coeff in coefficients[::-1]:
        binary_str += str(coeff)

    return binary_str

poly_str = "x^3 + x^2 + 1"
```

```
# binary_representation = polynomial_to_binary(poly_str)
print(f"Binary representation of '{poly_str}': {polynomial_to_binary(poly_str)}")
```

```
     Binary representation of 'x^3 + x^2 + 1': 1101
```

```
import random

def xor(a, b):
    result = []
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else:
            result.append('1')
    return ''.join(result)

def xordiv(dividend, divisor, sender=False):
    if sender:
        dividend += '0' * (len(divisor) - 1)

    pick = len(divisor)
    tmp = dividend[0: pick]

    while pick < len(dividend):
        if tmp[0] == '1':
            tmp = xor(divisor, tmp) + dividend[pick]
        else:
            tmp = xor('0' * pick, tmp) + dividend[pick]
        pick += 1

    if tmp[0] == '1':
        tmp = xor(divisor, tmp)
    else:
        tmp = xor('0' * pick, tmp)

    checkword = tmp
    return checkword

def encodeData(data, key):
    remainder = xordiv(data, key, True)
    return data + remainder

def print_changed_bit(original_data, received_data):
    for i in range(len(original_data)):
        if original_data[i] != received_data[i]:
            print(f"Bit {i + 1} changed: Original bit = {original_data[i]}, Received bit = {received_data[i]}")

def correct_data(original_data, received_data):
    corrected_data_list = list(received_data)
    for i in range(len(original_data)):
        if original_data[i] != received_data[i]:
            corrected_data_list[i] = original_data[i]
    corrected_data = ''.join(corrected_data_list)
    return corrected_data

def simulate_correct_received_data(data, key):
    encoded_data = encodeData(data, key)
    print("Encoded Data:", encoded_data)

    # No error introduced, data is valid
    rem = xordiv(encoded_data, key)
    if int(rem) == 0:
        print("Valid Data")
        #print("Received Data:", data)
    else:
        print("Invalid Data")

def simulate_error_received_data(data, key):
    encoded_data = encodeData(data, key)
    print("Encoded Data:", encoded_data)

    # Simulate an error by flipping a random bit
    error_position = random.randint(0, len(encoded_data) - 1)
    encoded_data_list = list(encoded_data)
    original_bit = encoded_data_list[error_position]
    encoded_data_list[error_position] = '1' if original_bit == '0' else '0'
    corrupted_data = ''.join(encoded_data_list)
    print("Received Data with Error : ", corrupted_data)

    # Print which bit is changed
    print_changed_bit(data, corrupted_data)
    # Correct the data
    corrected_data = correct_data(data, corrupted_data)
    print("Corrected Data:", corrected_data)

    # Check the received data for errors
    rem = xordiv(corrected_data, key)
    if int(rem) == 0:
        print("Data Corrected and Valid")
    else:
        print("Data Not Corrected and Invalid")

data = "1011001100"
key = polynomial_to_binary(poly_str)

print("Sender Side----------------------------")
print("Data:", data)
print("Key:", key)
print("CRC:", xordiv(data, key, True))
print()

print("Receiver Side (Correct Data)-------------------------")
simulate_correct_received_data(data, key)

print("Receiver Side (Error Data)-------------------------")
simulate_error_received_data(data, key)
```

```
     Sender Side----------------------------
     Data: 1011001100
     Key: 1101
     CRC: 111

     Receiver Side (Correct Data)-------------------------
     Encoded Data: 1011001100111
     Valid Data
     Receiver Side (Error Data)-------------------------
     Encoded Data: 1011001100111
     Received Data with Error :  1011011100111
     Bit 6 changed: Original bit = 0, Received bit = 1
     Corrected Data: 1011001100111
     Data Corrected and Valid
```

```
def bit_stuffing(data):
    stuffed_data = ""
    count = 0

    for bit in data:
        if bit == '1':
            count += 1
            stuffed_data += bit
        else:
            count = 0
            stuffed_data += bit

        if count == 5:
            stuffed_data += '0'
            count = 0

    return stuffed_data

def bit_destuffing(stuffed_data):
    destuffed_data = ""
    count = 0

    for bit in stuffed_data:
        if bit == '1':
            count += 1
            destuffed_data += bit
        else:
            destuffed_data += bit

        if count == 5:
            # Skip the next bit, which is the stuffed '0'
            count = 0

    return destuffed_data

if __name__ == "__main__":
    data = "0111111101111011101111"

    print("Original Data: ", data)

    stuffed_data = bit_stuffing(data)
    print("Stuffed Data: ", stuffed_data)

    destuffed_data = bit_destuffing(stuffed_data)
    print("Destuffed Data: ", destuffed_data)
```

```
Original Data:  0111111101111011101111
Stuffed Data:  01111101101111011101111
Destuffed Data:  01111101101111011101111
```

```
FLAG = "01111110"
ESC = "00000000"

def byte_stuffing(data):
    umpp = {}
    print("Data before byte stuffing:")
    print(data)

    freq = len(data) // 8
    i = 0
    while i < freq:
        if data.find(FLAG) >= 0 and data.find(FLAG) < len(data) and not umpp.get(data.find(FLAG), False):
            ind = data.find(FLAG)
            umpp[ind] = True
            umpp[ind + 8] = True
            data = byte_stuff(data, ind)

        if data.find(ESC) >= 0 and data.find(FLAG) < len(data) and not umpp.get(data.find(ESC), False):
            ind = data.find(ESC)
            umpp[ind] = True
            umpp[ind + 8] = True
            data = byte_stuff(data, ind)

        i += 1

    print("Data after byte stuffing:", data)

def byte_stuff(data, index):
    temp = data[index:]  # last part
    data = data[:index]  # first part
    data += ESC  # appending in first and last middle
    data += temp
    return data

data = "01010101000010100111111001010100000000"
byte_stuffing(data)
```

```
Data before byte stuffing:
01010101000010100111111001010100000000
Data after byte stuffing: 01010101000010100000000000000000000000000111111001010100000000
```

```
FLAG = 'FLAG'
ESC = 'ESC'

def byte_stuffing(data):
    umpp = {}
    print("Data before byte stuffing:")
    print(data)

    i = 0
    while i < len(data):
        if data[i:i+len(FLAG)] == FLAG and not umpp.get(i, False):
            umpp[i] = True
            umpp[i + len(FLAG)] = True
            data = byte_stuff(data, i, FLAG)
            i += len(FLAG) - 1

        if data[i:i+len(ESC)] == ESC and not umpp.get(i, False):
            umpp[i] = True
            umpp[i + len(ESC)] = True
            data = byte_stuff(data, i, ESC)
            i += len(ESC) - 1

        i += 1

    print("Data after byte stuffing:", data)

def byte_stuff(data, index, replace_with):
    temp = data[index:]  # last part
    data = data[:index]  # first part
    data += replace_with  # appending in the first and last middle
    data += temp
    return data

data = "AFLAGBCESCDEFLAGF"
byte_stuffing(data)
```

```
Data before byte stuffing:
AFLAGBCESCDEFLAGF
Data after byte stuffing: AFLAGFLAGBCESCESCDEFLAGFLAGF
```