Name: Ahaan Desai
Roll no.: 231070016
Batch: A

Computer Network Lab
Experiment 5

Aim: To Implement Bit and Byte stuffing

Theory

Bit stuffing and byte stuffing are data transparency techniques used at the data link layer in communication protocols. Their main purpose is to ensure that special control sequences used to indicate the start and end of a data frame are not mistakenly found within the actual data being transmitted.

1. **Bit Stuffing**: Bit stuffing is the process of inserting extra bits into the data stream to ensure that the flag pattern used for framing is not misinterpreted as part of the data. To avoid confusion between actual data and control flags (like frame start/end delimiters), especially in bit-oriented protocols such as HDLC (High-Level Data Link Control).

Working:

- The frame delimiter used is typically the flag byte `01111110` (in HDLC).
- To prevent this pattern from appearing in the data, the transmitter inserts a `0` after every sequence of five consecutive `1`s in the data stream.
- The receiver removes the stuffed `0` after every five `1`s to recover the original data.

2. **Byte Stuffing**: Byte stuffing is the process of inserting special bytes (characters) into the data to distinguish between control characters and actual data. This is mainly used in byte-oriented protocols.

The following control characters are used:

- STX (Start of Text) → Indicates the start of the frame
- ETX (End of Text) → Indicates the end of the frame
- DLE (Data Link Escape) → Used to escape control characters in the data

If the data contains any control characters like STX, ETX, or DLE, the receiver may misinterpret them as framing delimiters. To avoid this, DLE is used as a prefix, and the character following DLE is interpreted specially.

How It Works:

- When the sender encounters STX, ETX, or DLE in the data:
  - It inserts a DLE byte before the control character.
  - This is known as escaping the character.
- The receiver recognizes DLE and treats the next byte as data, not as a control character.

Program

```python
STX = 0x02
ETX = 0x03
DLE = 0x10

def bit_stuffing(data):
    stuffed = ""
    consecutive_ones = 0

    for ch in data:
```

```
        # Iterate over bits of ascii character
        ch_binary = bin(ord(ch))[2:].zfill(8)
        for bit in ch_binary:
            stuffed += bit

            if bit == '1':
                consecutive_ones += 1

                if consecutive_ones == 5:
                    stuffed += "0"
                    consecutive_ones = 0
            else:
                consecutive_ones = 0
    return stuffed

def bit_unstuffing(data):
    ascii_output = ""
    bit_buffer = ""
    one_count = 0

    for ch in data:
        if ch == '0':
            if one_count != 5:
                bit_buffer += ch

            one_count = 0
        else:
            bit_buffer += ch
            one_count += 1

        if len(bit_buffer) == 8:
            ascii_output += chr(int(bit_buffer, 2))
            bit_buffer = ""

    return ascii_output

def byte_stuffing(data):
    stuffed = chr(STX)

    for ch in data:
```

```
        if ch in [chr(DLE), chr(STX), chr(ETX)]:
            stuffed += chr(DLE)
        stuffed += ch

    stuffed += chr(ETX)

    return stuffed


def byte_unstuffing(data):
    original = ""
    i = 1

    while i < len(data) - 1:
        if data[i] == chr(DLE):
            i += 1
            original += data[i]
        else:
            original += data[i]
        i += 1

    return original

string = "abcdesodghlkjsgsbgjklsbljfg"
encoded = bit_stuffing(string)
print("Bit stuffing encoded string:", encoded)
decoded = bit_unstuffing(encoded)
print("Bit stuffing decoded string:", decoded)
assert decoded == string

print()

string = "Hello World!"
string += chr(DLE)
string += " Hello Python!"
encoded = byte_stuffing(string)
print("Byte stuffing encoded string:", repr(encoded))
decoded = byte_unstuffing(encoded)
print("Byte stuffing decoded string:", decoded)
assert decoded == string
```

Output

```
(.venv) sysadmin@sysadmin:~/New Folder 1$ python bit_stuffing.py
Bit Stuffing:
Input string:  abcdesodghlkjsgsbgjklsbljfg
Bit stuffing encoded string: 011000010110001001100011011001000110010101110011011010
11011000111001101100010011011000110101001100110011001111
Bit stuffing decoded string: abcdesodghlkjsgsbgjklsbljfg

Byte Stuffing:
Input string:  'Hello World!\x10 Hello Python!'
Byte stuffing encoded string: '\x02Hello World!\x10\x10 Hello Python!\x03'
Byte stuffing decoded string: 'Hello World!\x10 Hello Python!'
(.venv) sysadmin@sysadmin:~/New Folder 1$
```

Conclusion

The experiment successfully demonstrated bit and byte stuffing techniques, ensuring reliable data framing by preventing misinterpretation of control sequences within data. Bit stuffing handles bit-level transparency, while byte stuffing uses escape characters for byte-level control. Both methods enhance data integrity and synchronization in communication protocols.