

**Activity based**

**Project Report on**

**Computer Networks**

**Submitted to Vishwakarma University, Pune**

**Under the Initiative of**

**Contemporary Curriculum, Pedagogy, and Practice (C2P2)**

**By**

**Merwin Pinto**

**SRN No : 202100102**

**Roll No : 01**

**Diya Oswal**

**SRN No : 202101718**

**Roll No : 40**

**Div : E**

**Third Year Engineering**

**Department of Computer Engineering**

**Faculty of Science and Technology**

**Academic Year**

**2023-2024**

**Project Description:**

Parity Bit Checker for Error detection and Correction:

The parity bit checker is a network method designed to detect errors and check the integrity of the data received at the receiver side by the sender side. The parity check method adds a bit to the original data for checking errors at the receiver end.

There are mainly two types of Parity that is **Even Parity and Odd Parity**

**This Parity Checker is further divided into 3 parts**

1. Message representation into Binary bits and frames
2. Parity bit checking using Even and odd parity concepts
3. comparing performances with other error detection Techniques

**PROJECT MODULE 1 :**

**Message representation into Binary bits and frames**

Message representation into binary bits and frames is a crucial process in digital communication, enabling the transmission of data over a communication channel. This process involves converting text messages into a sequence of binary bits, which are then structured into frames for efficient transmission.

**1. Character Encoding**

The first step involves converting the message characters into their binary representation using a character encoding scheme, such as ASCII or Unicode. These schemes assign a unique binary code to each character, ensuring consistent representation across different systems.

**2. Framing**

Once the message is encoded in binary, it is divided into frames, which are structured units of data tailored for transmission. Each frame typically consists of a header, payload, and error detection/correction codes.

**Example**

Suppose we want to send the message "Hello, World!" using ASCII encoding. The ASCII codes for each character are:

H = 01001000

e = 01100101

l = 01101100

l = 01101100

o = 01101111

, = 00101100

Space = 00100000

W = 01010111

o = 01101111

r = 01110010

l = 01101100

d = 01100100

! = 00100001

Combining the ASCII codes for each character, we get the binary representation of the message:

0100100001100101011011000110110001101111001011000010000001010111011011110111001001110010011011000110010000100001

This binary representation can be divided into frames, each with a header and payload. The header might contain information about the frame type, frame number, and sequence number. The payload would contain the actual data being transmitted.

**Flowchart**

The following flowchart illustrates the step-by-step process of the Frame\_creation function:

**Start**

**|**

**Convert characters to binary**

**|**

**Create frames**

**|**

**Add header, payload, and trailer**

**|**

**Add parity bits**

**|**

**Return frames**

**|**

**End**

**Implementation :**

Implementation Procedure

**Step 1: Convert the string to binary**

We first convert each letter of the input string to its binary representation and then converting the ASCII code to binary using the bin() function. The zfill() method is used to pad the binary representation of each letter with zeros to the specified char\_bit length.

**Step 2: Create frames**

The function then creates a list of frames, where each frame is a string of binary digits representing a single packet of data. The counter variable is used to keep track of the number of bits that have been added to the current frame.

**Step 3: Create the Header and Trailer**

The function creates the Header and Trailer by converting the sender's MAC address and the receiver's MAC address to binary and concatenating them together. The Payload is created by converting the input string to binary using the text\_to\_binary() function.

**Code :**

def Frame\_creation(string,frame\_size,char\_bit):

  binary\_letters = []

  for letter in string:

    binary\_letter = bin(ord(letter))[2:].zfill(char\_bit)

    binary\_letters.append(binary\_letter)

  print("The letters converted to binary binary\_letters",binary\_letters)

  frames = []

  counter = 0

  frame = ""

  for binary\_letter in binary\_letters:

    frame += binary\_letter

    counter += char\_bit

    if counter == frame\_size:

      frames.append(frame)

      counter = 0

      frame = ""

  if counter > 0:

    frame += "0" \* (frame\_size - counter)

    frames.append(frame)

  print(frames)

  bit = 48

  binr = bin(bit)

  binary\_string = binr[2:]

  sender\_MAC =   binary\_string

  receiver\_mac = binary\_string

  Header = str(sender\_MAC) + str(receiver\_MAC)

  print("The Message converted to binary ")

  Payload = str(text\_to\_binary(string))

  Trailer = str(frames)

  arr=[]

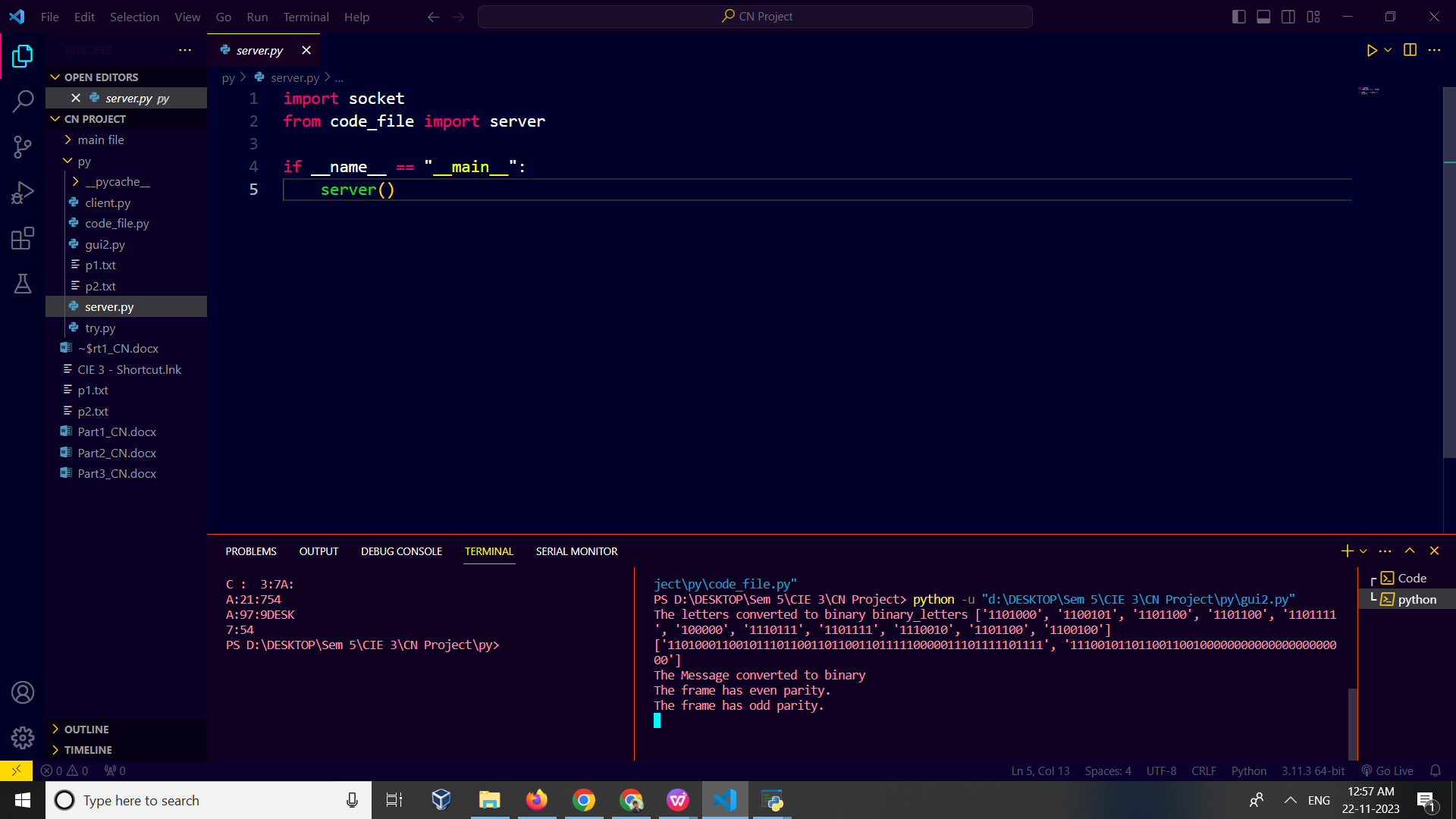
  for frame in frames:

    frame = Header + Payload + Trailer

    arr.append(frame)

  return frames

**Output:**



**Conclusion :**

In summary, the Frame creation effectively prepared the input text for transmission by converting it to binary, dividing it into frames, and adding necessary header and trailer information.