JADAVPUR UNIVERSITY

Faculty of Engineering & Technology

....CSE/PC/B/S/314 Computer Networks Lab...Engg. Laboratory

Name.....JEETESH ABROL.....

Class....CSE, UG3... Sec.....A1..... Roll No...002210501021....

Date of Experiment..... Date of Submission.....

Marks Obtained..... Signature of Examiner.....

NAME	CO-WORKER	ROLL
.....Jeetesh Abrol.....		....002210501021...
.....		.....
.....		.....
.....		.....
.....		.....
.....		.....
.....		.....
.....		.....
.....		.....
.....		.....
.....		.....

Experiment No....Assignment 1 .....

Commence at..... Completed at.....

Name of Teacher concerned PROF. SARBANI ROY

TITLE: *Design and implement an error detection module which has two schemes namely Checksum and Cyclic Redundancy Check (CRC).*

OBJECT:

Test the above two schemes for the error types and CRC polynomials mentioned above for the following cases (not limited to).

- Error is detected by both CRC and Checksum.
- Error is detected by checksum but not by CRC.
- Error is detected by CRC but not by Checksum.

DESIGN

- The module is written in python.

- It consists of five .py header files:
  1. helper.py — Defines the different CRC divisor polynomials
  2. crc.py — Implements CRC Error Detection
  3. checksum.py — Implements Checksum Error Detection
  4. client.py — Client program that sends the data and receives if the data accepted or not
  5. server.py — Server program that establishes connection

- The server and client program implement a **connection-oriented server** using **TCP/IP** protocol, bymaking use of sockets in python.

- Server binds on port **3000** and waits for client to send a packet. The packet consists of the codeword,along with information about the encoding scheme used.

- The server runs in an *infinite loop* and forks off a separate process for each connection request. Hence, all the sender requests are serviced by the same receiver program.

- Server reads the contents. Then, it converts the data into a **string of 0sand 1s**. This string is then divided into frames depending on PKT_SIZE.

- The frames are then encoded using each of the different Error Detection schemes, thus generating multiple codewords.

- Then an error is generated and the **same error** is injected in each of the codewords.

- Then the server sends each packet to the receiver via a connection, and reads the **verdict** of the receiver in return. **0**- No error (PASS), **1**- Error detected (FAIL).

IMPLEMENTATION

helper.py

```
from sympy import Poly, symbols


# Define the variable x
x = symbols('x')


CRCPolynomials = {
    "CRC_1"       : "x+1",
    "CRC_4_ITU"   : "x^4 + x^1 + 1",
    "CRC_5_ITU"   : "x^5 + x^4 + x^2 + 1",
    "CRC_5_USB"   : "x^5 + x^2 + 1",
    "CRC_6_ITU"   : "x^6 + x^1 + 1",
    "CRC_7"       : "x^7 + x^3 + 1",
    "CRC_8_ATM"   : "x^8 + x^2 + x^1 + 1",
    "CRC_8_CCITT" : "x^8 + x^7 + x^3 + x^2 + 1",
    "CRC_8_MAXIM" : "x^8 + x^5 + x^4 + 1",
    "CRC_8"       : "x^8 + x^7 + x^6 + x^4 + x^2 + 1",
    "CRC_8_SAE"   : "x^8 + x^4 + x^3 + x^2 + 1",
    "CRC_10"      : "x^10 + x^9 + x^5 + x^4 + x^1 + 1",
    "CRC_12"      : "x^12 + x^11 + x^3 + x^2 + x + 1",
    "CRC_15_CAN"  : "x^15 + x^14 + x^10 + x^8 + x^7 + x^4 + x^3 + 1",
    "CRC-16"      : "x^16 + x^15 + x^2 + 1"
}


def convToBinary(key):
    # Get the polynomial expression
    polynomial_expr = CRCPolynomials[key]

    # Create a polynomial object
    polynomial = Poly(polynomial_expr, x)

    # Get the coefficients of the polynomial
    coeffs = polynomial.all_coeffs()

    # Convert coefficients to binary representation
    degree = len(coeffs) - 1
    binary = ''.join('1' if i < len(coeffs) and coeffs[i] != 0 else '0' for i in
range(degree, -1, -1))

    return binary
```

checksum.py

```
class Checksum:
    @classmethod
    def generate_checksum(cls, chunks):
        res = ""
        size = len(chunks[0])
        for chunk in chunks:
            res = bin(int(res, 2)+int(chunk, 2)) if res != "" else chunk
        res = res[2:]
        res = bin(int(res[-size:], 2)+int(res[:-size], 2))
        return ''.join('1' if x == '0' else '0' for x in res[2:])


    @classmethod
    def check_checksum(cls, chunks, checksum):
        res = ""
        size = len(chunks[0])
        for chunk in chunks:
            if chunk == '':
                continue
            res = bin(int(res, 2)+int(chunk, 2)) if res != "" else chunk
        res = res[2:]
        res = bin(int(res[-size:], 2)+int(res[:-size], 2)+int(checksum, 2))
        if res.count("1") == size:
            return True
        else:
            return False
```

crc.h

```
#!/usr/bin/env python3
class CRC:
    @classmethod
    def xor(cls, 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)

    @classmethod
    def mod2div(cls, dividend, divisor):
```

```
        pick = len(divisor)


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


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


        checkword = tmp
        return checkword


    @classmethod
    def encodeData(cls, chunks, key):
        l_key = len(key)
        enc_chunks = []
        for data in chunks:
            appended_data = data + '0'*(l_key-1)
            remainder = cls.mod2div(appended_data, key)
            codeword = data + remainder
            enc_chunks.append(codeword)
        return enc_chunks


    @classmethod
    def checkRemainder(cls, chunks, key):
        l_key = len(key)
        for data in chunks:
            appended_data = data + '0'*(l_key-1)
            remainder = cls.mod2div(appended_data, key)
            if remainder.count('1') != 0:
                return False


        return True
```

server.py

```
import socket
from Crypto.Util.number import long_to_bytes, bytes_to_long
import checksum
import crc


s = socket.socket()
s.bind(('localhost', 3000))
s.listen(1)
print("Waiting for connections")


while True:
    c, addr = s.accept()
    print("Connection from {}".format(addr))
    for i in range (2):
        try:
            size = bytes_to_long(c.recv(1024))
            print(f"Data size received: {size}")


            method = c.recv(1024).decode('utf-8')
            print(f"Detection method received: {method}")


            li = []


            if method == "CRC":

                divisor = c.recv(1024).decode('utf-8')
                print(f"CRC divisor received: {divisor}")



                while True:
                    text = c.recv(1024).decode('utf-8')
                    print(text + "\n")
                    if 'EOF' in text:
                        li.append(text.replace("EOF", ""))
                        print("end")
                        break
                    li.append(text)


                received_text = ''.join(li)[:size]
                print("Received Data (CRC)={}".format(received_text))
```

```
    if crc.CRC.checkRemainder(li, divisor):

                c.send(b"Received Text, No errors found by CRC")
            else:
                c.send(b"Error detected by CRC")


        elif method == "Checksum":


            checksum_value = c.recv(1024).decode('utf-8')
            print(f"Checksum value received: {checksum_value}")


            while True:
                text = c.recv(1024).decode('utf-8')
                if 'EOF' in text:
                    li.append(text.replace("EOF", ""))
                    break
                li.append(text)


            received_text = ''.join(li)[:size]
            print("Received Data (Checksum)={}".format(received_text))


            if checksum.Checksum.check_checksum(li, checksum_value):
                c.send(b"Received Text, No errors found by Checksum")
            else:
                c.send(b"Error detected by Checksum")


        else:
            raise Exception("Invalid detection method received")


    except Exception as e:
        print(f"Error: {e}")
```

client.py

```
import socket
from Crypto.Util.number import bytes_to_long, long_to_bytes
import random
import time
import checksum
import crc
import helper
```

```
HOST = 'localhost'
PORT = 3000
c = socket.socket()
random.seed(time.time())


def inject_error(text: str, number: int) -> str:
    if number == 0:
        return text
    for _ in range(number):
        x = random.randint(0, len(text) - 1)
        if text[x] == '0':
            text = text[:x] + '1' + text[x + 1:]
        else:
            text = text[:x] + '0' + text[x + 1:]
    return text


def inject_error_2(text: str , num) -> str:
    #CRC_1 is x+1 and by inserting 11 at the end the codeword becomes divisible by x+1
and thus CRC is not able to detect the error
    modified_text = '0' * (len(text) - 2) + "11"
    return modified_text



c.connect((HOST, PORT))


text = input("Enter data:").encode('utf-8')
res = input("Do you want to insert errors?(Y/n)")
enc_text = bin(bytes_to_long(text))[2:]
actual_len = len(enc_text)


if actual_len % 8 != 0:
    extra = '0' * (8 - (actual_len % 8))
    enc_text += extra


c.send(long_to_bytes(actual_len))
time.sleep(1)

print("CRC Method In Progress")
c.send(b"CRC")
time.sleep(1)
crc_method = input("Give the CRC divisor method:")
divisor = helper.convToBinary(crc_method)
c.send(divisor.encode('utf-8'))
```

```
PKT_SIZE_CRC = 8


# crc_chunks = []
# for i in range(0, len(enc_text), PKT_SIZE_CRC):
#     crc_chunks = enc_text[i:i + PKT_SIZE_CRC]


crc_chunks = [enc_text[i:i + PKT_SIZE_CRC] for i in range(0, len(enc_text),
PKT_SIZE_CRC)]
crc_encoded_chunks = crc.CRC.encodeData(crc_chunks, divisor)


for chunk in crc_encoded_chunks:
    time.sleep(1)
    if res.lower() == 'y':
        injected_chunk = inject_error(chunk, random.randint(0, 2)) #if inject_error_2
crc1 doesn't work
        c.send(injected_chunk.encode('utf-8'))
    else:
        c.send(chunk.encode('utf-8'))


c.send(b'EOF')
crc_result = c.recv(1024).decode('utf-8')
print(f"CRC Result: {crc_result}")


time.sleep(2)


c.send(long_to_bytes(actual_len))
print("Checksum Method In Progress")
c.send(b"Checksum")
time.sleep(1)


PKT_SIZE_CHECKSUM = 16
swap_flag = False   #if it is True then checksum will not work


checksum_chunks = [enc_text[i:i + PKT_SIZE_CHECKSUM] for i in range(0, len(enc_text),
PKT_SIZE_CHECKSUM)]
checksum_value = checksum.Checksum.generate_checksum(checksum_chunks)
c.send(checksum_value.encode('utf-8'))
if swap_flag and len(checksum_chunks) > 1:
    checksum_chunks[0], checksum_chunks[-1] = checksum_chunks[-1], checksum_chunks[0]
```

```
for chunk in checksum_chunks:
    time.sleep(1)
    if not swap_flag and res.lower() == 'y':
        injected_chunk = inject_error(chunk, random.randint(0, 2))
        c.send(injected_chunk.encode('utf-8'))
    else:
        c.send(chunk.encode('utf-8'))
c.send(b'EOF')


checksum_result = c.recv(1024).decode('utf-8')
print(f"Checksum Result: {checksum_result}")


c.send(b'EOF')
c.close()
```

When no error is inserted:

```
jeet@jeet-vivobook:~/Dropbox/CN ass1$ python client.py
Enter data:hello everyone
Do you want to insert errors?(Y/n)n
CRC Method In Progress
Give the CRC divisor method:CRC_1
CRC Result: Received Text, No errors found by CRC
Checksum Method In Progress
Checksum Result: Received Text, No errors found by Checksum
```

Burst Error not divisible by CRC is inserted:

```
jeet@jeet-vivobook:~/Dropbox/CN ass1$ python client.py
Enter data:hello everyone
Do you want to insert errors?(Y/n)y
CRC Method In Progress
Give the CRC divisor method:CRC_1
CRC Result: Error detected by CRC
Checksum Method In Progress
Checksum Result: Received Text, No errors found by Checksum
```

Burst Error divisible by CRC polynomial:

```
jeet@jeet-vivobook:~/Dropbox/CN ass1$ python client.py
Enter data:hello everyone
Do you want to insert errors?(Y/n)Y
CRC Method In Progress
Give the CRC divisor method:CRC_1
CRC Result: Received Text, No errors found by CRC
Checksum Method In Progress
Checksum Result: Error detected by Checksum
```

Single or Double error divisible by CRC polynomial:

```
jeet@jeet-vivobook:~/Dropbox/CN ass1$ python client.py
Enter data:hello everyone
Do you want to insert errors?(Y/n)Y
CRC Method In Progress
Give the CRC divisor method:CRC_1
CRC Result: Error detected by CRC
Checksum Method In Progress
Checksum Result: Error detected by Checksum
```

On testing different data, we can analyse that:

- Checksum is the weakest error detection method out of all the others.
- The efficiency of CRC methods increases as the length of the generating polynomial increases.
- For the same number of redundant bits, CRC-8 outperforms 8-bit Checksum.
- CRC-32 is completely **immune** to all errors spanning **not more than 32 bits** in length. However, it is susceptible to errors spanning more than 32 bits.

COMMENTS

This assignment has helped me in understanding the different error detection schemes immensely, by researching and implementing them. It has also helped in understanding the demerits of a detection scheme, and how such demerits are overcome by other detection scheme.