Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Основы компьютерных сетей

ОТЧЕТ

по лабораторной работе №3

на тему

МЕТОДЫ ЗАЩИТЫ ОТ ОШИБОК ИСПОЛЬЗУЕМЫЕ В СЕТЯХ

Студент Т.Ю. Петрович

Преподаватель В.А. Марцинкевич

МИНСК 2024

**1 ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ**

**1.1 Цель работы**

Изучить методы защиты от ошибок, применяемые в СПД. Отработать программы, реализующие процедуры формирования помехозащитных кадров и получения информации из них.

**1.2 Исходные данные к работе**

Требования к наполнению программы:

Реализовать поддержку поля FCS в структуре кадра - для проверки кадра с помощью циклического кода. При этом циклический код применять только к полю Data. Длину поля FCS необходимо рассчитать с учетом исходных требований (в битах, поскольку циклический код имеет битовую природу) и «округлить» (до байтов, поскольку минимальной единицей передаваемых через COM-порт данных является байт).

Исходные требования к циклическому коду: код должен обнаруживать и исправлять одиночные ошибки.

В рамках кодирования и декодирования кадров, программно реализовать алгоритм деления полиномов (делить как «на бумаге»).

Предусмотреть возможность случайного искажения одного случайного бита в одном случайном байте в поле Data каждого кадра после приема. Вероятность искажения должна составлять 70 %.

Модифицировать окно состояния. По-прежнему периодически выводить структуру текущего кадра после приема, но немного по-другому (до декодирования). Один кадр по-прежнему должен соответствовать одной строке. При этом выделять (подчеркиванием либо другим цветом) поле FCS вместо принятого значения поля FCS выводить вычисленное.

**2 ОБЗОР ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ**

Циклические коды относятся к числу блоковых систематических кодов, в которых каждая комбинация кодируется самостоятельно (в виде блока) таким образом, что информационные (m) и контрольные (k) символы всегда находятся на определенных местах.

Возможность обнаружения и исправления практически любых ошибок при относительно малой избыточности по сравнению с другими кодами, а также простота схемной реализации аппаратуры кодирования и декодирования сделали эти коды широко распространенными.

В основу циклического кодирования положено использование неприводимого многочлена P(х), который применительно к циклическим кодам называется образующим, генераторным или порождающим многочленом (полиномом).

Неприводимые многочлены нельзя представить в виде произведения многочленов низших степеней. Они играют роль, сходную с простыми числами в теории чисел. Неприводимые полиномы P(x) можно записать в виде десятичных или двоичных чисел либо в виде алгебраического многочлена (полинома).

Циклический код определяется с помощью порождающего полинома P(х) степени К. Посредством операции над полиномом, выполняемой с участием P(x) определяемым m битами сообщения, подлежащего передаче, образуется так называемый кодовый полином, который делится без остатка на P(x). Этот кодовый полином передается вместо исходного сообщения. Если при передаче в нем произошла ошибка, то он делиться без остатка не будет. Следовательно, по значению остатка можно судить о наличии ошибки в принятом коде.

Алгоритм применения циклического кода:

1. Выбрать число k контрольных разрядов по формуле.

2. Выполнить побитовый сдвиг исходного сообщения P(x) на k разрядов влево.

3. Разделить полученное число P1(x) на образующий полином степени k.

4. Остаток от деления R(x) сложить с P1(x) – полученная комбинация является итоговой.

Идея обнаружения ошибок в принятом коде заключается в том, что при отсутствии ошибок закодированная комбинация делится на порождающий многочлен без остатка.

Алгоритм проверки на ошибки полученного сообщения:

1. Разделить сообщение на образующий полином.

2. Если остаток от деления равен 0, то ошибки отсутствуют.

3. Если остаток ненулевой, подчитывается вес остатка W. Если вес равен или меньше числа исправляемых ошибок S, то принятый код складывают по модулю 2 с кодом остатка R(x) и получают исправленную комбинацию.

4. Если W> S, проводится циклический сдвиг на один разряд принятой посылки и полученный код снова делится на образующий полином и так, пока W не станет меньше или равен S. Далее циклически сдвинутую комбинацию складывают с остатком и циклически сдвигают вправо столько раз, сколько было сдвинуто влево.

**3 ВЫПОЛНЕНИЕ РАБОТЫ**

**3.1 Обоснование выбранной системы программирования**

Для эмуляции COM-портов в данной работе была выбрана программа Eterlogic Virtual Serial Ports Emulator.

Эта программа предназначена для создания виртуальных COM-портов, которые можно использовать для тестирования и отладки программного обеспечения, взаимодействующего с серийными портами.

Основные особенности и преимущества Eterlogic Virtual Serial Ports Emulator включают:

- Создание виртуальных портов – программа позволяет создавать пары виртуальных COM-портов, которые могут обмениваться данными между собой так же, как физические порты.

- Поддержка различных приложений – эмулятор совместим с большинством программ, которые работают с COM-портами, что делает его полезным инструментом для разработки и тестирования.

Для взаимодействия с COM-портами в программе была использована библиотека pySerial.

Она предоставляет удобный интерфейс для работы с последовательными портами на языке Python, также она поддерживает такие функции, как открытие порта, отправка и приём данных, настройка параметров порта, таких как скорость передачи данных, количество стоп-битов и битов чётности.

Библиотека Tkinter использовалась для создания графического пользовательского интерфейса (GUI) в Python. Она предоставляет инструменты для создания оконных приложений с современным и адаптивным дизайном.

**3.2 Разработка программного обеспечения**

Для реализации данной задачи былм написаны функции для работы с алгоритмом циклического кода:

func (p CRCProtector) CalculateCRC(data string) string – метод для расчета циклического избыточного кода (CRC) для переданных данных.

func (p CRCProtector) CheckCRCWithSyndrome(data\_with\_crc string, fcs string) string – метод для проверки данных с CRC и исправления возможных ошибок на основе синдрома ошибок.

Также были написаны вспомогательные функции:

func bits\_to\_string(bits []int) string – функция для преобразования списка бит в строку.

func corrupt\_bit\_with\_probability(data string) string – функция случайного искажения одного бита с вероятностью ошибки.

func poly\_division\_mod2(dividend []int, divisor []int) ([]int, []int) – функция деления двоичных чисел по модулю 2, возвращающая частное и остаток.

func generate\_syndrome\_map(polynomial []int, data\_length int) map[[]int]int – функция генерации карты синдромов для обнаружения позиций ошибок.

**4 РЕЗУЛЬТАТЫ РАБОТЫ**

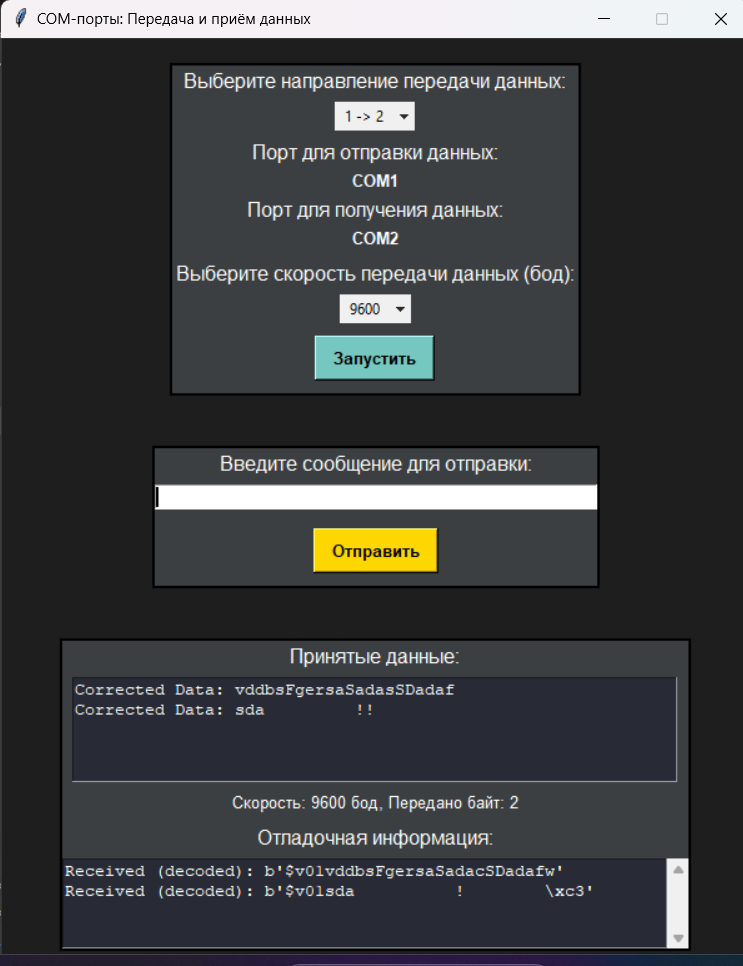
****

Рисунок 4.1 – Пример работы программы

**5 ВЫВОДЫ**

В ходе выполнения лабораторной работы были изучены методы защиты от ошибок, применяемые в СПД.

Был реализован модуль кодирования данных, реализующий процедуры формирования помехозащитных кадров и получения информации из них.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

[1] pySerial - pySerial 3.4 documentation [Электронный ресурс]. Режим доступа: https://pyserial.readthedocs.io/en/latest/pyserial.html. – Дата доступа: 25.09.2024.

[2] tkinter – Python interface to Tcl/Tk Python 3.12.6 documenatation [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/tkinter.html. – Дата доступа: 25.09.2024.

[3] Вычислительные комплексы, системы и сети: лабораторный практикум для студентов специальности I-40 02 01 «Вычислительные машины, системы и сети» / И. И. Глецевич, Д. В. Ламовский, Д. А.Пынькин. – Минск : БГУИР, 2010. – 36 с. : ил.

# **ПРИЛОЖЕНИЕ А**

(обязательное)

Исходный текст программы

Содержимое файла crc.py:

import random

polynomial = [1, 1, 0, 0, 0, 1, 0, 1, 1]

def bits\_to\_string(bits):

    chars = []

    for i in range(0, len(bits), 8):

        byte\_bits = bits[i:i + 8]

        byte\_str = ''.join(str(bit) for bit in byte\_bits)

        char = chr(int(byte\_str, 2))

        chars.append(char)

    return ''.join(chars)

def corrupt\_bit\_with\_probability(data):

    data\_bytes = bytearray(data.encode('utf-8'))

    if len(data\_bytes) == 0:

        return data

    byte\_index = random.randint(0, len(data\_bytes) - 1)

    bit\_index = random.randint(0, 7)

    if random.random() < 0.7:

        print("123123123123")

        data\_bytes[byte\_index] ^= (1 << bit\_index)

    return data\_bytes.decode('utf-8')

def poly\_division\_mod2(dividend, divisor):

    deg\_dividend = len(dividend) - 1

    deg\_divisor = len(divisor) - 1

    if deg\_dividend < deg\_divisor:

        return [0], dividend

    quotient = [0] \* (deg\_dividend - deg\_divisor + 1)

    remainder = dividend[:]

    for i in range(deg\_dividend - deg\_divisor + 1):

        if remainder[i] == 1:

            quotient[i] = 1

            for j in range(len(divisor)):

                remainder[i + j] ^= divisor[j]

    while len(remainder) > 0 and remainder[0] == 0:

        remainder.pop(0)

    return quotient, remainder

def calculate\_crc(data):

    global polynomial

    bits = []

    for char in data:

        bin\_representation = bin(ord(char))[2:].zfill(8)

        bits.extend([int(bit) for bit in bin\_representation])

    bits.extend([0, 0, 0, 0, 0, 0, 0, 0])

    quotient, remainder = poly\_division\_mod2(bits, polynomial)

    byte = 0

    for bit in remainder:

        byte = (byte << 1) | bit

    return chr(byte)

def generate\_syndrome\_map(polynomial, data\_length):

    syndrome\_map = {}

    for i in range(data\_length):

        bits = [0] \* data\_length

        bits[i] = 1

        \_, remainder = poly\_division\_mod2(bits, polynomial)

        syndrome\_map[tuple(remainder)] = i

    return syndrome\_map

def check\_crc\_with\_syndrome(data\_with\_crc, fcs):

    global polynomial

    bits = []

    for byte in data\_with\_crc:

        if isinstance(byte, str):

            byte = ord(byte)

        bin\_representation = format(byte, '08b')

        bits.extend([int(bit) for bit in bin\_representation])

    if isinstance(fcs, str):

        fcs = ord(fcs)

    fcs\_bits = [int(bit) for bit in format(fcs, '08b')]

    bits.extend(fcs\_bits)

    total\_bits = len(bits)

    syndrome\_map = generate\_syndrome\_map(polynomial, total\_bits)

    quotient, remainder = poly\_division\_mod2(bits, polynomial)

    if remainder == [0] \* len(remainder):

        return data\_with\_crc

    else:

        syndrome = tuple(remainder)

        if syndrome in syndrome\_map:

            error\_bit\_position = syndrome\_map[syndrome]

            print(f"Обнаружена ошибка в бите {error\_bit\_position}, исправляем")

            bits[error\_bit\_position] ^= 1

            print(bits\_to\_string(bits[:len(bits) - 8]))

            return bits\_to\_string(bits[:len(bits) - 8])

        else:

            print("Ошибка не может быть исправлена.")

            return data\_with\_crc