1. Министерство высшего образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 9**

«Методы контроля целостности»

1. по дисциплине «Основы Информационной Безопасности»
2. Выполнил
3. студент гр. Белоконь Д. А.
4. <*подпись*>

Проверил Пахомов М. А.

1. <*подпись*>

1. Санкт-Петербург
2. 2023
3. **Цель работы**

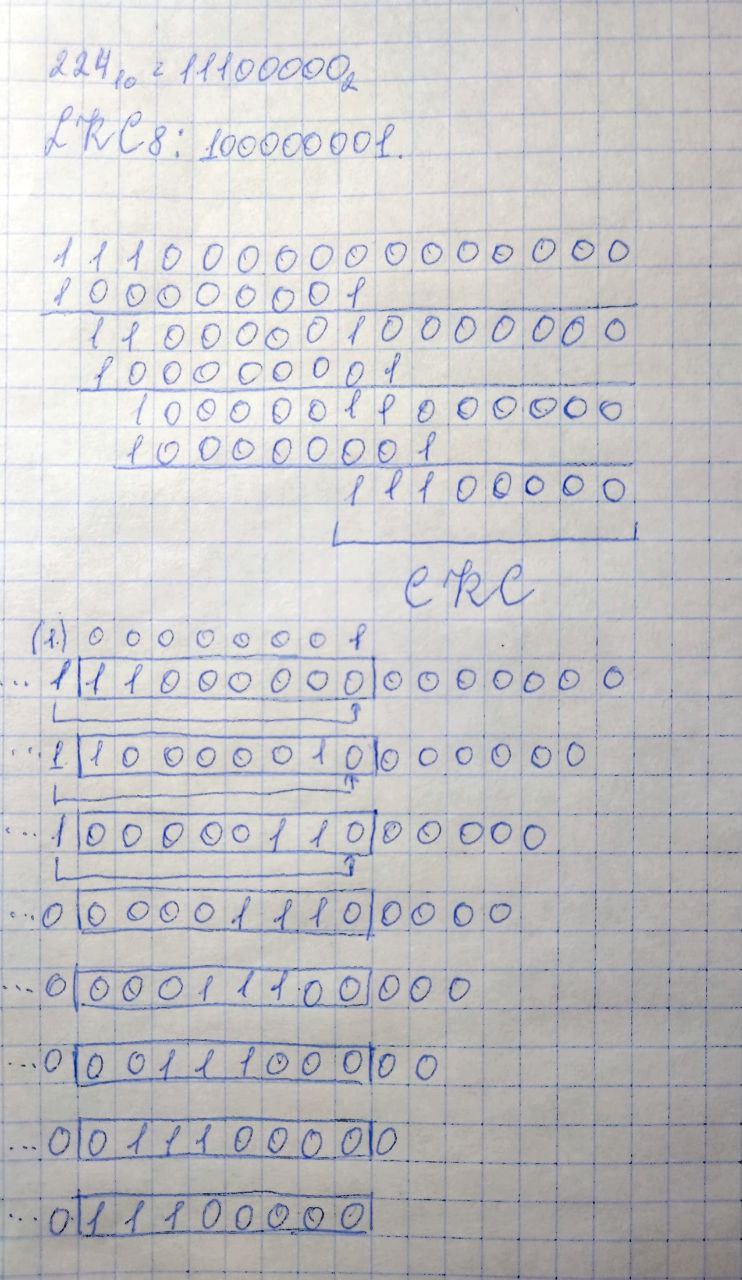
Приобретение навыков в разработке и использовании программных средств контроля целостности информации, анализ свойств алгоритмов расчёта контрольных сумм и циклических кодов.

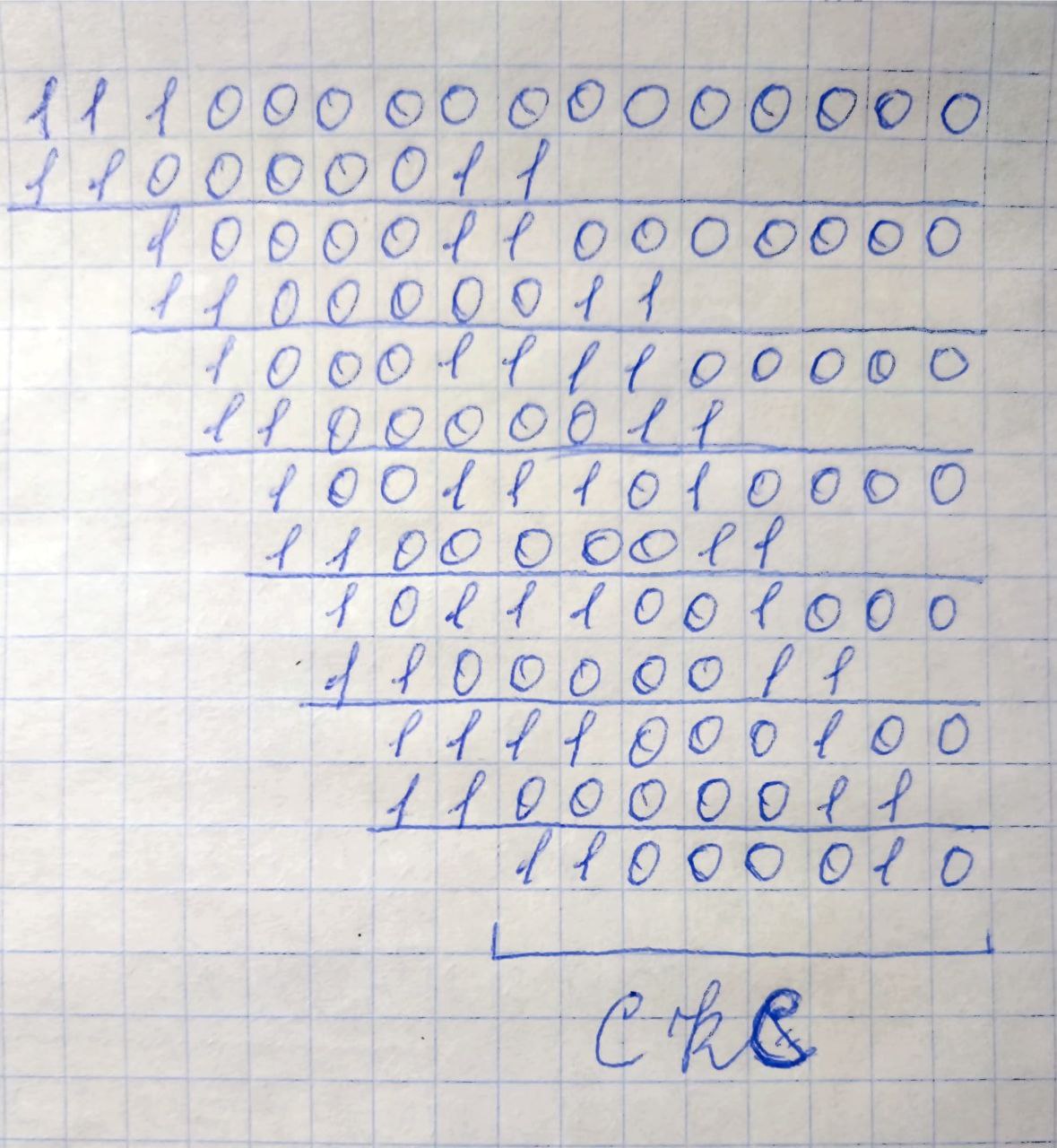
Формулировка задания

Необходимо вручную подсчитать CRC для случайного двоичного числа длиной 8 бит. Затем нужно разработать вычисляющую двумя способами — делением и сдвигами — CRC программу и сравнить результаты её работы для различных исходных и модифицированных данных.

Результаты

Для первого шага возьмём число 224, которое в двоичном представлении записывается как 11100000. Затем для LRC8 посчитаем CRC: добавим 8 пустых нулей в конец числа и найдём остаток от деления на 100000001. Получается 11100000, то есть изначальное число. Для сдвиговых операций так же. При расчёте CRC для полинома (8, 7, 1, 0) результатом является 11000010.

Рис 1. Расчёт LRC8 двумя методам

Рис 2. Расчёт CRC для полинома (8, 7, 1, 0)

Далее напишем простой алгоритм, высчитывающий контрольную сумму через сложение числовых значений всех символов по модулю 256. Для тестирования добавим Python алгоритм, генерирующий 100 случайных символов и записывающий их в файл input.txt.

Протестируем. При запуске программы результатом является число 255. Затем заменим последнюю большую M на N. Логично, что результат становится равным (255 + 1) % 256 = 0. Если дублировать строку, то число изменяется на (255 + 255) % 256 = 254. От перемены мест слагаемых сумма не меняется — 255.

Далее напишем универсальный алгоритм, высчитывающий CRC для произвольного файла по заданному полиному длиной 8 через полиномиальное деление. Для этого будем представлять данные с помощью массива чисел 1 и 0. В качестве стандартного полинома впишем 100000001.

Пересчитаем CRC для той же строки, получаем 35 (00100011). Произведя те же преобразования, что и простым алгоритмом, получаем: 32 (00100000), 0 (00000000), 35 (00100011).

После добавления двух байтовых нулей (16 бинарных) в начало файла итоговое число остаётся равным 35 (00100011). Это объясняется тем, что сначала порождающий полином через XOR встраивает себя самого на место нулей, а затем, заново удаляя, идёт по прежнему алгоритму.

Если же добавить в нужный текст полученный CRC, то при пересчёте получится, логично, ноль. Этот результат означает, что контрольная сумма была посчитана правильно. Однако в случае повторения кода снова получается значение, равное CRC. Это является закономерным, ибо любой символ, вписанный после CRC, станет новой контрольной суммой, так как после получения нуля регистр сразу сдвигается на оставшийся символ. При добавлении байта 80h результат становится равным 163 (10100011). Обычная контрольная сумма, что логично, так как данный байт не является CRC.

Далее работаем с бинарными файлами. Скачаем «Бесов» Достоевского и вычислим контрольную сумму — 32 (00100000). Добавим её дважды в конец файла, получим, естественно, тот же CRC.

Проводя исследование на существование контрольных сумм, не подтверждающих достоверность информации, не находим подобных значений. Приходим к выводу, что алгоритм работает отлично.

И наконец, изменив алгоритм расчёта CRC на бинарные сдвиговые операции, я получил иной результат, который, тем не менее, даёт ноль при подстановке в конец файла.

Вывод

Лабораторная вышла достаточно трудоёмкой из-за необходимости отслеживать большое количество преобразований в процессе работы. Кроме того, приходилось даже переписывать алгоритм ради более оптимальной работы: в прошлой версии программы файл сначала полностью подгружался в память, а затем уже обрабатывался. В конечном счёте, всё свелось к простой загрузке байтов в ограниченный массив и постоянному применению операции XOR.

Теоретические знания полезны только на этом же уровне, поскольку в современных системах на низком уровне всё уже достаточно давно слаженно работает.

**Приложение**

Приложение 1. Подпрограммы расчёта CRC.

char \*moveCRC(char \*fileName, int polynom) {

char \*polynomBin = itob(polynom);

FILE \*inputFile = fopen(fileName, "r");

if (inputFile == NULL) {

fprintf(stderr, "Could not open input file");

return NULL;

}

char \*buffer = (char \*)malloc(sizeof(char) \* 16);

for (int i = 0; i < 16; i++)

buffer[i] = -1;

int counter = 0;

unsigned char symbol = 'a', \*binary;

char isLoaded = 0;

while ("This shit gets too long") {

char last = buffer[0];

memcpy(buffer, buffer + 1, 15);

buffer[15] = -1;

if (buffer[8] == -1) {

if ((symbol = fgetc(inputFile)) == 255 && feof(inputFile)) {

if (isLoaded) {

break;

}

isLoaded += 1;

loadSymbol(0, buffer, 16);

continue;

}

loadSymbol(symbol, buffer, 16);

continue;

}

if (last) {

char \*binary = xor(buffer, polynomBin, 9);

memcpy(buffer, binary, 9);

free(binary);

}

}

free(polynomBin);

fclose(inputFile);

char \*result = (char \*)malloc(sizeof(char \*) \* 8);

for (int i = 0; i < 8; i++)

result[i] = 0;

int i = -1;

counter = 7;

while (buffer[++i] != -1);

for (i -= 1;i > -1; i--) {

result[counter--] = buffer[i];

}

free(buffer);

return result;

}

char \*divisionCRC(char \*fileName, int polynom) {

char \*polynomBin = itob(polynom);

FILE \*inputFile = fopen(fileName, "r");

if (inputFile == NULL) {

fprintf(stderr, "Could not open input file");

return NULL;

}

char \*buffer = (char \*)malloc(sizeof(char) \* 16);

for (int i = 0; i < 16; i++)

buffer[i] = -1;

int counter = 0;

unsigned char symbol = 'a', \*binary;

char isLoaded = 0;

while ("This shit gets too long") {

if (buffer[8] == -1) {

if ((symbol = fgetc(inputFile)) == 255 && feof(inputFile)) {

if (isLoaded) {

break;

}

isLoaded += 1;

loadSymbol(0, buffer, 16);

continue;

}

loadSymbol(symbol, buffer, 16);

continue;

}

char \*binary = xor(buffer, polynomBin, 9);

memcpy(buffer, binary, 9);

free(binary);

updateBinary(buffer, 16);

}

free(polynomBin);

fclose(inputFile);

char \*result = (char \*)malloc(sizeof(char \*) \* 8);

for (int i = 0; i < 8; i++)

result[i] = 0;

int i = -1;

counter = 7;

while (buffer[++i] != -1);

for (i -= 1;i > -1; i--) {

result[counter--] = buffer[i];

}

free(buffer);

return result;

}

int main(int argc, char \*argv[]) {

if (argc < 2) {

fprintf(stderr, "No input file specified");

return -1;

}

FILE \*inputFile = fopen(argv[1], "r");

if (inputFile == NULL) {

fprintf(stderr, "Could not open input file");

return -2;

}

unsigned char controlSum;

char symbol;

while ((symbol = fgetc(inputFile)) != EOF) {

controlSum += symbol;

}

fclose(inputFile);

printf("Control sum for the file is: %d\n", controlSum);

return 0;

}