****Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Группа: М80-207Б-18

Студент: Ильминский Никита Сергеевич

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2019.

***Постановка задачи***Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы (Windows/Unix).

*Вариант 23: Коды Хэмминга.*

Необходимо реализовать 2 программы:

Первая программа кодирует по Хэммингу нормальную последовательность байт. Вторая программа раскодирует и корректирует закодированную последовательность (корректировка осуществляется только, если было повреждение).

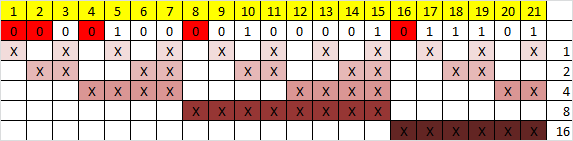
Многопоточность необходимо реализовать только в программе 2 (дешифрование и коррекция кодов).

Количество контрольных разрядов в соображениях упрощения задачи фиксировано (используется Код Хэмминга (12,8), то есть 8 значащих бит и 4 контрольных бита).

***Алгоритм***

Для выполнения данной задачи необходимо создать две программы.

Первая программа будет считывать сообщение с входного потока и сохранять его в двоичном представлении; далее полученный двоичный код будет делиться на блоки по 8 бит, и для каждого блока будет происходить кодирование: на места, индекс которых равен степеням двойки, вставляются сначала нулевые контрольные биты, затем их значение вычисляется на основе тех разрядов, которые “контролирует” (или “покрывает”) каждый контрольный бит: контрольный бит с номером N контролирует все последующие N бит через каждые N бит, начиная с позиции N (то есть для контрольного бита 1 это будут разряды с номерами 1,3,5,7,9,11,13,15,..., для второго 2,3,6,7,10,11,14,15,..., для четвертого 4,5,6,7,12,13,14,15,20,21,22,23,..., и так далее), данная закономерность отображена на картинке:



Чтобы вычислить значение для данного контрольного бита, нужно вычислить XOR сумму (проверить на четность количество единиц) всех контролируемых им разрядов, если сумма равна 0, то значение контрольного бита будет равно 0, и наоборот. На этом первая часть алгоритма завершается.

Вторая программа будет выполнять проверку полученного кода на ошибки, их исправление и наконец декодирование, получение исходного сообщения: для проверки программа должна заново вычислять контрольные биты и сравнивать их с полученным сообщением, сумма индексов несовпадающих контрольных битов будет равна индексу разряда, в котором произошла ошибка. Инвертировав поврежденный бит и отбросив все контрольные биты, мы получим исходное сообщение (важно заметить, что код Хэмминга способен исправлять ошибку лишь в одном разряде, поэтому тесты с двумя ошибками в одном 12-битном блоке будут выдавать непредсказуемый результат).

***Листинг***

*12-8Ham.c*

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

//Hamming (12,8)

#define TOTAL\_BITS 12

#define DATA\_BITS 8

int power(int base, int exp) {

int num = 1;

while(exp != 0) {

num \*= base;

--exp;

}

return num;

}

char\* encode(const char\* msg) {

size\_t pBits = TOTAL\_BITS - DATA\_BITS;

size\_t blockNum = strlen(msg) / DATA\_BITS;

char\* code = malloc(blockNum \* TOTAL\_BITS + 1);

code[blockNum \* TOTAL\_BITS] = '\0';

strcpy(code, msg);

size\_t pBitIdx;

size\_t blockShift;

size\_t nextBlock;

for (size\_t i = 0; i < blockNum; i++) {

pBitIdx = 1;

blockShift = TOTAL\_BITS \* i;

for (size\_t n = 0; n < pBits; n++) {

for (size\_t j = strlen(code); j != pBitIdx + blockShift - 1; j--) {

code[j] = code[j - 1];

}

code[pBitIdx + blockShift - 1] = '0';

pBitIdx \*= 2;

}

pBitIdx = 1;

for (size\_t n = 0; n < pBits; n++) {

nextBlock = TOTAL\_BITS \* (i + 1);

int sum = 0;

for (size\_t j = blockShift; j < nextBlock; j++) {

if (((j - blockShift + 1) & 1 << n) == power(2, n)) {

sum ^= code[j] - '0';

}

}

sum == 0 ? (code[pBitIdx + blockShift - 1] = '0') : (code[pBitIdx + blockShift - 1] = '1');

pBitIdx \*= 2;

}

}

return code;

}

int main() {

char msg[64];

printf("Hamming code:\nEnter the message you want to encode (no more than 63 chars):\n");

fgets(msg, 64, stdin);

size\_t msgLen = strlen(msg) - 1;

size\_t binMsgSize = msgLen \* 8 + 1;

char\* binMsg = malloc(binMsgSize);

size\_t n = 0;

for(size\_t i = 0; i < msgLen; i++) {

for(size\_t j = 8; j != 0; j--)

(msg[i] & 1 << j - 1) ? (binMsg[n++] = '1') : (binMsg[n++] = '0');

}

binMsg[msgLen \* 8] = '\0';

printf("Input binary message:\n%s\n", binMsg);

printf("Hamming (12,8) encoding...\n");

char\* code = encode(binMsg);

printf("Encoded message:\n%s\n", code);

free(binMsg);

free(code);

return 0;

}

***Вывод***

В результате выполнения данной лабораторной работы я получил первый опыт работы с потоками в -nix системе.

Потоки связаны с процессами, и они могут показаться похожими (и те и те являются независимыми единицами выполнения), однако они имеют существенные различия: каждый процесс выполняется в отдельном адресном пространстве, каждому процессу выделяются свои отдельные ресурсы, а взаимодействие процессов реализуется при помощи дополнительных инструментов (например, pipe), в то время как потоки используют общую область памяти, имеют собственные регистры и стек, доступ к которым могут получать другие потоки.

работа с потоками осуществляется при помощи системных вызовов библиотеки pthread.h:

pthread\_create – создание потока и выполнение необходимой функции (задается в аргументах к функции);

pthread\_exit – завершение работы потока;

pthread\_join – ожидание выполнения необходимого потока; при помощи этой функции можно синхронизировать потоки между собой (помогает решить проблему того, что потоки в программе часто могут выполняться не в очереди их создания).

Эти знания понадобятся мне при дальнейшей работе с ОС.