Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий
институт
Кафедра «Информатика»
кафедра

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Лабораторная работа №4. Управление файлами в ОС GNU/Linux
тема

Преподаватель _____ А.С. Кузнецов
подпись, дата инициалы, фамилия

Студент КИ18-17/1б 031830504 _____ Е.В. Железкин
номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2019

СОДЕРЖАНИЕ

1 Цель работы

Программная реализация обработки текстовой информации, хранящейся во внешней памяти, с использованием системных вызовов низкоуровневого ввода-вывода.

2 Краткие теоретические сведения

make — утилита, автоматизирующая процесс преобразования файлов из одной формы в другую. Чаще всего это компиляция исходного кода в объектные файлы и последующая компоновка в исполняемые файлы или библиотеки.

Функция **open(char *pathname, int oflag, int pmode)** открывает файл, определяемый по path - имени, и подготавливает его к последующему чтению или записи, что определяется посредством oflag.

Функция **close(int handle)** относится к UNIX-подобной системе и не определяется стандартом ANSI C. При вызове функции **close** с действительным дескриптором файла она закрывает связанный с ним файл, осуществив предварительно очистку буфера записи, если это необходимо. (Дескрипто-ры файлов создаются при успешном обращении к open() или creat() и не относятся к потокам или указателям на файлы.)

Функция **write(int handle, char *buffer, unsigned int count)** записывает байты из буфера в файл. Операции **write** начинаются с текущей позиции указателя на файл (указатель ассоциирован с заданным файлом). Если файл открыт для добавления, операции выполняются в конец файла. После осуществления операций записи указатель на файл (если он есть) увеличивается на количество действительно записанных байтов.

Функция **read(int handle, char *buffer)** делает попытку считать заданное число байт из файла, связанного с handle, в буфер, адресуемый параметром buffer.

Функция **remove(const char *filename)** удаляет файл или каталог. Удаляемый каталог должен быть пустым, иначе он удален не будет. Так же у программы должны быть права доступ к файлу (каталогу) разрешающие удаление файла (каталога).

fstat(int fd, struct stat *statbuff) не определена стандартом ANSI C. Функция **fstat** заполняет структуру, на кото-рую указывает statbuf, информацией о файле, связанном с дескриптором файла fd. Информация о содержимом stat может быть найдена в файле sys\stat.h.

Функция **ftruncate(int fd, off_t length)** устанавливают длину обычного файла с файловым дескриптором fd в length байт. Если файл до этой операции был длиннее, то отсеченные данные теряются. Если файл был короче, то он увеличивается, а добавленная часть заполняется нулевыми байтами.

Указатель на файл не меняется.

3 Задача работы

Требуется разработать программу в виде Linux-приложения, позволяющую считывать, модифицировать существующую и добавлять новые структуры (записи) фиксированной длины из/во входные и выходные файлы. Обязательны к реализации функции по добавлению одной записи, модификации записи (значения всех полей или только части из них), удаления записи, чтения одной записи, отображения всех записей (или части). Должен использоваться интерфейс командной строки (CLI). Общее описание структуры данных и два дополнительных запроса данных, обязательных к реализации, даются индивидуально и приводятся далее.

Вариант 12. Структура данных: кафедра; количество преподавателей; количество профессоров. Создать два запроса, позволяющих определить кафедры, где нет профессоров, и кафедры, в которых их доля максимальна.

4 Ход работы

Листинг 1 – содержание файла functions.h проекта SP_4:

```
#ifndef SP_4_FUNCTIONS_H
#define SP_4_FUNCTIONS_H

#include <unistd.h>
#include "structs.h"

/*! \brief Функция, обрабатывающая ввод.
 *  @param strForCheck Строка считанная с клавиатуры.
 *  @return number Число, содержащееся в строке, код ошибки(-1) в противном случае.
 */
int InputChecker(char* strForCheck);



/*! \brief Функция, обрабатывающая работу функций read для цикла(для удобства).
 *  @param returnFromRead возвращаемое функцией read значение.
 *  @return Код выполнения функции.
 */
int CheckRead(int returnFromRead);



/*! \brief Функция, ищущая структуру в файле по номеру.
 *  @param argvMain Параметры для записи в файл.
 *  @param fdMain Дескриптор файла.
 *  @return Код завершения функции.
 */
int FilePlaceFinder(int fid, int number);



/*! \brief Функция, выводящее меню помощи.
 */
void ShowHelp();

#endif //SP_4_FUNCTIONS_H
```

Листинг 2 – содержание файла structs.h проекта SP_4:

```
#ifndef SP_4_STRUCTS_H
#define SP_4_STRUCTS_H
#define MAX_LENGTH 50

/*! \struct department
 *  \brief Структура, отвечающая за хранение кафедр.
 */
struct department
{
    /*! Название кафдеры. */
    char name[MAX_LENGTH];
```

```
    /*! Число учителей. */
    double countOfTeachers;
    /*! Число профессоров. */
    double countOfProfessors;
};

#endif //SP_4_STRUCTS_H
```

Листинг 3 – содержание файла FileFunctions.h проекта SP_4:

```
#ifndef SP_4_FILEFUNCTIONS_H
#define SP_4_FILEFUNCTIONS_H

#include "structs.h"

/*! \brief Функция, записывающая структуру в файл.
 *  @param argvMain Параметры для записи в файл.
 *  @param fdMain Дескриптор файла.
 *  @return Код завершения функции.
 */
int AddElementInFile (char** argvMain, int fdMain);


/*! \brief Функция, очищающая файл.
 *  @param fdMain Дескриптор файла.
 *  @return Код завершения функции.
 */
int DeleteAllFromFile(int fid);


/*! \brief Функция, изменяющая поля структуры в файле.
 *  @param argvMain Параметры для записи в файл.
 *  @param fdMain Дескриптор файла.
 *  @return Код завершения функции.
 */
int ModifyElementInFile(char** argvMain, int fdMain);


/*! \brief Функция, выводящая структуру по номеру из файла.
 *  @param argvMain Параметры для записи в файл.
 *  @param fdMain Дескриптор файла.
 *  @return Код завершения функции.
 */
int ShowInfoFromFileByNumber(char** argvMain, int fdMain);


/*! \brief Функция, выводящая все данные из файла.
 *  @param fdMain Дескриптор файла.
 *  @return Код завершения функции.
 */
int ShowAllFromFile(int fidMain);


/*! \brief Функция, удаляющая элемент структуры из файла.
 *  @param argvMain Параметры для записи в файл.
 *  @param fdMain Дескриптор файла.
 *  @return Код завершения функции.
 */
int DeleteElementFromFile(char** argvMain, int fdMain);
```

```
/*! \brief Функция, выполняющая первый запрос.
 *  @param fdMain Дескриптор файла.
 *  @return Код завершения функции.
 */
int FirstRequest(int fdMain);


/*! \brief Функция, выполняющая второй запрос.
 *  @param fdMain Дескриптор файла.
 *  @return Код завершения функции.
 */
int SecondRequest(int fdMain);

#endif //SP_4_FILEFUNCTIONS_H
```

Листинг 4 – содержание файла main.c проекта SP_4:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "functions.h"
#include "FileFunctions.h"
#define MAX_LENGTH 50
#define ERROR_CODE -1
#define SUCCESS 0
#define CHOOSE_HELP 1
#define CHOOSE_ADD 2
#define CHOOSE_SHOW_ONE 3
#define CHOOSE_SHOW_ALL 4
#define CHOOSE_MODIFY 5
#define CHOOSE_DELETE_ONE 6
#define CHOOSE_DELETE_ALL 7
#define CHOOSE_FIRST_REQUEST 8
#define CHOOSE_SECOND_REQUEST 9
#define CHOOSE_EXIT 10

#define UNKNOWN_INPUT_MASSAGE "Для вывод меню импользуйте параметр \"1\"!\n"

/*! \brief Главная функция.
 *  @return Код завершения программы.
 */
int main(int argc, char* argv[])
{

    unsigned int number = 0;

    char* path = "Save";
    mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH;
    int fid = open(path, O_RDWR | O_CREAT, mode);
    if (fid == ERROR_CODE)
    {
        printf("Не удалось открыть файл!\n");
        return 0;
    }
```

```
if (argv[1] == NULL)
{
    printf("Отсутствуют входные данные!\n");
    return 0;
}
number = atoi(argv[1]);

switch (number)
{
case CHOOSE_HELP:
    ShowHelp();
    break;
case CHOOSE_ADD:
    if (AddElementInFile(argv, fid) == ERROR_CODE)
    {
        printf("Не удалось добавить элемент в файл!\n");
        return ERROR_CODE;
    }
    break;
case CHOOSE_SHOW_ONE:
    if (ShowInfoFromFileByNumber(argv, fid) == ERROR_CODE)
    {
        printf("Не удалось вывести элемент!\n");
        return ERROR_CODE;
    }
    break;
case CHOOSE_SHOW_ALL:
    if (ShowAllFromFile(fid) == ERROR_CODE)
    {
        printf("Не удалось вывести элементы!\n");
        return ERROR_CODE;
    }
    break;
case CHOOSE_MODIFY:
    if (ModifyElementInFile(argv, fid) == ERROR_CODE)
    {
        printf("Не удалось изменить элемент!\n");
        return ERROR_CODE;
    }
    break;
case CHOOSE_DELETE_ONE:
    if (DeleteElementFromFile(argv, fid) == ERROR_CODE)
    {
        printf("Не удалось Удалить элемент!\n");
        return ERROR_CODE;
    }
    break;
case CHOOSE_DELETE_ALL:
    if (DeleteAllFromFile(fid) == ERROR_CODE)
    {
        printf("Не удалось очистить файл!\n");
        return ERROR_CODE;
    }
    break;
case CHOOSE_FIRST_REQUEST:
    if (FirstRequest(fid) == ERROR_CODE)
    {
        printf("Не удалось выполнить запрос!\n");
        return ERROR_CODE;
    }
```

```
            break;
        case CHOOSE_SECOND_REQUEST:
            if (SecondRequest(fid) == ERROR_CODE)
            {
                printf("Не удалось выполнить запрос!\n");
                return ERROR_CODE;
            }
                break;
        case CHOOSE_EXIT:
            close(fid);
            return SUCCESS;
        default:
            printf(UNKNOWN_INPUT_MASSAGE);
            close(fid);
            return SUCCESS;
        }

        close(fid);
        return SUCCESS;

}
```

Листинг 4 – содержание файла functions.c проекта SP_4:

```
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "structs.h"

#define MAX_LENGTH 50

#define ERROR_CODE -1

#define SUCCESS 0

#define ASCII_CONST 48

#define NEXT_DIGIT 10

#define ERROR_READ_OF_FILE "Ошибка чтения файла!\n"

#define MENU_MASSAGE "Меню:\n"\
            "\"1\" - Вывести информацию о командах;\n"\
            "\"2\" - Добавить новую кафедру в файл(Интерфейс: 2 "\
            "\"Название\" \"Количество профессоров\" \""\
            "Количество преподавателей\");\n"\
            "\"3\" - Вывести информацию о кафедре из файла по её номеру"\
            "(Интерфейс: 3 \"Номер выводимой кафедры\");\n"\
            "\"4\" - Вывести информацию о всех кафедрах из файла"\
            "(Интерфейс: 4);\n"\
            "\"5\" - Изменить информацию о кафедре(Интерфейс: 4 \"Порядковый"\
            " номер кафедры\"\n\"1|2|3 - соответствующие поля(название; "\
            "количество профессоров; количество преподавателей)\" \""\
            "Устанавливаемое значение\");\n"\
            "\"6\" - Удалить кафедру из файла по её номеру(Интерфейс: 6 \"Номер
удаляемой кафедры\");\n"\
```

```
            "\"7\" - Очистить файл(Интерфейс: 7)\n"\
            "\"8\" - Запрос, позволяющий определить кафедры, на которых "\
            "нет профессоров(Интерфейс: 8);\n"\
            "\"9\" - Запрос, позволяющий определить кафедру с наибольшим "\
            "процентом профессоров на ней(Интерфейс: 9);\n"\
            "\"10\" - Выход(Интерфейс: 10).\n"

int InputChecker(char* strForCheck)
{
    int number = 0;

    for (unsigned long i = 0; i < strlen(strForCheck); i++)
    {
        if (strForCheck[i] == '0' || strForCheck[i] == '1' ||
            strForCheck[i] == '2' || strForCheck[i] == '3' ||
            strForCheck[i] == '4' || strForCheck[i] == '5' ||
            strForCheck[i] == '6' || strForCheck[i] == '7' ||
            strForCheck[i] == '8' || strForCheck[i] == '9') number =
                                                                number * NEXT_DIGIT
+ (strForCheck[i] - ASCII_CONST);
        else
            return ERROR_CODE;
    }

    return number;
}

int CheckRead(int returnFromRead)
{
    if (returnFromRead == 0) return SUCCESS;
    if (returnFromRead == ERROR_CODE)
    {
        return SUCCESS;
    }
    return 1;
}

int FilePlaceFinder(int fdMain, int number)
{
    unsigned int size;
    int executionCode = 0;
    struct department temp;

    executionCode = lseek(fdMain, 0, SEEK_SET);

    if (executionCode == ERROR_CODE)
    {
        printf(ERROR_READ_OF_FILE);
        return ERROR_CODE;
    }

    executionCode = read(fdMain, &size, sizeof (unsigned int));
    if (executionCode == ERROR_CODE || executionCode != sizeof (unsigned int))
    {
        printf(ERROR_READ_OF_FILE);
        return ERROR_CODE;
    }

    struct stat fileInfo;
    executionCode = fstat(fdMain, &fileInfo);
```

```
    if (executionCode == ERROR_CODE)
    {
        printf(ERROR_READ_OF_FILE);
        return ERROR_CODE;
    }

    if (fileInfo.st_size - sizeof (unsigned int) < size * number)
        return ERROR_CODE;

    for (int i = 0; i < number - 1; i++)
        if (read(fdMain, &temp, size) != size)
         return ERROR_CODE;

    return size;
}

void ShowHelp()
{
    printf(MENU_MASSAGE);
}
```

Листинг 4 – содержание файла FileFunctions.c проекта SP_4:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include "structs.h"
#include "functions.h"

#define MAX_LENGTH 50

#define ERROR_CODE -1

#define SUCCESS 0

#define ASCII_CONST 48

#define NEXT_DIGIT 10

#define CHANGE_NAME 1

#define CHANGE_COUNT_OF_PROFESSORS 2

#define CHANGE_COUNT_OF_TEACHERS 3

#define INFINITY 999999999

#define FIRST_PARAM 2

#define SECOND_PARAM 3

#define THIRD_PARAM 4

#define LAST_ONE 1

#define ERROR_INPUT_IS_EMPTY "Отсутствуют входные данные!\n"
```

```
#define ERROR_NAME_IS_TOO_LONG "Слишком длинное название!\n"

#define ERROR_OF_INCORRECT_INPUT "Некорректные входные данные!\n"

#define ERROR_OF_INCORRECT_COUNT_OF_TEACHERS "Некорректное количество
преподавателей!\n"

#define ERROR_OUTPUT_IN_FILE "Ошибка записи элемента структуры!\n"

#define ERROR_OUTPUT_ELEMENT_IN_FILE "Ошибка записи элемента структуры!"

#define ERROR_READ_FROM_FILE "Ошибка считывания из файла!\n"

#define ERROR_INCORRECT_NUMBER_OF_ELEMENT "Некорректный номер элемента!\n"

#define ERROR_CONFLICT_T_P "Профессоров не должно быть больше, чем преподавателей!\n"

#define TEMP_FILE_NAME "tempSave"

#define ERROR_INCORRECT_NUMBER_OF_DEPARTMENT "Некорректный номер кафедры!\n"

#define ERROR_DEPARTMENTS_IS_MISSING "Кафедры отсутствуют!\n"

#define SECOND_REQUEST_MASSAGE "Кафедры, на которых доля профессоров максимальна:\n"

int AddElementInFile (char** argvMain, int fdMain)
{
    int executionCode = 0;
    unsigned int size = sizeof (struct department);
    struct department temp;

    if (argvMain[FIRST_PARAM] == NULL || argvMain[SECOND_PARAM] == NULL ||
argvMain[THIRD_PARAM] == NULL)
    {
        printf(ERROR_INPUT_IS_EMPTY);
        return ERROR_CODE;
    }

    if (strlen(argvMain[FIRST_PARAM]) > MAX_LENGTH)
    {
        printf(ERROR_NAME_IS_TOO_LONG);
        return ERROR_CODE;
    }

    for (unsigned long i = 0; i < strlen(argvMain[FIRST_PARAM]) + LAST_ONE; i++)
        temp.name[i] = argvMain[FIRST_PARAM][i];

    if (atoi(argvMain[SECOND_PARAM]) != InputChecker(argvMain[SECOND_PARAM]) ||
        atoi(argvMain[THIRD_PARAM]) < atoi(argvMain[SECOND_PARAM]))
    {
        printf(ERROR_OF_INCORRECT_INPUT);
        return ERROR_CODE;
    }
    else
        temp.countOfProfessors = atoi(argvMain[SECOND_PARAM]);


    if (atoi(argvMain[THIRD_PARAM]) != InputChecker(argvMain[THIRD_PARAM]))
    {
```

```
        printf(ERROR_OF_INCORRECT_COUNT_OF_TEACHERS);
        return ERROR_CODE;
    }
    else
        temp.countOfTeachers = atoi(argvMain[THIRD_PARAM]);


    executionCode = lseek(fdMain, 0, SEEK_END);

    if (executionCode == ERROR_CODE)
    {
        printf(ERROR_OUTPUT_IN_FILE);
        return ERROR_CODE;
    }

    struct stat fileInfo;
    fstat(fdMain, &fileInfo);
    if (fileInfo.st_size == 0)
    {
        executionCode = write(fdMain, &size, sizeof (unsigned int));
        if (executionCode == ERROR_CODE)
        {
            printf(ERROR_OUTPUT_ELEMENT_IN_FILE);
            return ERROR_CODE;
        }
    }

    executionCode = write(fdMain, &temp, size);
    if (executionCode == ERROR_CODE)
    {
        printf(ERROR_OUTPUT_ELEMENT_IN_FILE);
        return ERROR_CODE;
    }

    return SUCCESS;

}

int DeleteAllFromFile(int fid)
{
    if (ftruncate(fid, 0) == ERROR_CODE)
        return ERROR_CODE;
    return SUCCESS;
}

int ModifyElementInFile(char** argvMain, int fdMain)
{
    if (argvMain[FIRST_PARAM] == NULL || argvMain[SECOND_PARAM] == NULL ||
argvMain[THIRD_PARAM] == NULL)
    {
        printf(ERROR_INPUT_IS_EMPTY);
        return ERROR_CODE;
    }

    int choose = 0;
    int number = 0;
    int size = 0;

    if (atoi(argvMain[FIRST_PARAM]) != InputChecker(argvMain[FIRST_PARAM]))
    {
```

```
    printf(ERROR_OF_INCORRECT_INPUT);
    return ERROR_CODE;
}
else
    number = InputChecker(argvMain[FIRST_PARAM]);

if (atoi(argvMain[SECOND_PARAM]) != InputChecker(argvMain[SECOND_PARAM]))
{
    printf(ERROR_OF_INCORRECT_INPUT);
    return ERROR_CODE;
}
else
    choose = InputChecker(argvMain[SECOND_PARAM]);

if ((choose == CHANGE_COUNT_OF_PROFESSORS || choose == CHANGE_COUNT_OF_TEACHERS) &&
    atoi(argvMain[THIRD_PARAM]) != InputChecker(argvMain[THIRD_PARAM]))
    {
        printf(ERROR_OF_INCORRECT_INPUT);
        return ERROR_CODE;
    }

if (lseek(fdMain, 0, SEEK_SET) == ERROR_CODE)
{
    printf(ERROR_READ_FROM_FILE);
    return ERROR_CODE;
}

int executionCode = read(fdMain, &size, sizeof (unsigned int));

if (executionCode == ERROR_CODE || executionCode < sizeof (unsigned int))
{
    printf(ERROR_READ_FROM_FILE);
    return ERROR_CODE;
}

struct department temp;

char* path = TEMP_FILE_NAME;
mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH;
int supFid = open(path, O_RDWR | O_CREAT, mode);

int count = 0;
DeleteAllFromFile(supFid);

while(CheckRead(read(fdMain, &temp, size)))
{
    if (write(supFid, &temp, size) == ERROR_CODE)
    {
        printf(ERROR_OUTPUT_IN_FILE);
        close(supFid);
        return ERROR_CODE;
    }

    count++;
}

if (InputChecker(argvMain[FIRST_PARAM]) > count)
{
    printf(ERROR_INCORRECT_NUMBER_OF_ELEMENT);
    close(supFid);
```

```
    return ERROR_CODE;
}

DeleteAllFromFile(fdMain);

if (lseek(fdMain, 0, SEEK_SET) == ERROR_CODE)
{
    printf(ERROR_READ_FROM_FILE);
    return ERROR_CODE;
}

if (lseek(supFid, 0, SEEK_SET) == ERROR_CODE)
{
    printf(ERROR_READ_FROM_FILE);
    return ERROR_CODE;
}

if (write(fdMain, &size, sizeof(unsigned int)) == ERROR_CODE)
{
    printf(ERROR_OUTPUT_IN_FILE);
    close(supFid);
    return ERROR_CODE;
}

for (int i = 1; i <= count; i++)
{
    executionCode = read(supFid, &temp, size);

    if (executionCode == ERROR_CODE || executionCode < sizeof (unsigned int))
    {
        printf(ERROR_READ_FROM_FILE);
        return ERROR_CODE;
    }

    if (number == i)
        switch (choose)
        {
        case CHANGE_NAME:
            strcpy(temp.name, argvMain[THIRD_PARAM]);
            break;
        case CHANGE_COUNT_OF_PROFESSORS:
            if (temp.countOfTeachers <
                InputChecker(argvMain[THIRD_PARAM]))
            {
                printf(ERROR_CONFLICT_T_P);
            }
            else
                temp.countOfProfessors =
                        InputChecker(argvMain[THIRD_PARAM]);
            break;
        case CHANGE_COUNT_OF_TEACHERS:
            if (temp.countOfProfessors >
                InputChecker(argvMain[THIRD_PARAM]))
            {
                printf(ERROR_CONFLICT_T_P);
            }
            else
                temp.countOfTeachers = InputChecker(argvMain[THIRD_PARAM]);
            break;
        }
```

```
        if (write(fdMain, &temp, size) == ERROR_CODE)
        {
            printf(ERROR_OUTPUT_IN_FILE);
            close(supFid);
            return ERROR_CODE;
        }
    }

    close(supFid);
    remove(TEMP_FILE_NAME);

    return SUCCESS;
}

int ShowInfoFromFileByNumber(char** argvMain, int fdMain)
{
    struct department temp;
    int size = 0;

    if (argvMain[FIRST_PARAM] == NULL)
    {
        printf(ERROR_INPUT_IS_EMPTY);
        return ERROR_CODE;
    }

    if (atoi(argvMain[FIRST_PARAM]) != InputChecker(argvMain[FIRST_PARAM]))
    {
        printf(ERROR_INCORRECT_NUMBER_OF_DEPARTMENT);
        return ERROR_CODE;
    }
    else
        size = FilePlaceFinder(fdMain, InputChecker(argvMain[FIRST_PARAM]));

    if (size == ERROR_CODE)
    {
        printf(ERROR_READ_FROM_FILE);
        return ERROR_CODE;
    }

    if (read(fdMain, &temp, size) == ERROR_CODE)
    {
        printf(ERROR_READ_FROM_FILE);
        return ERROR_CODE;
    }

    printf("Название: %s;  Количество профессоров: %g;  Количество преподавателей:
%g.\n",
           temp.name, temp.countOfProfessors, temp.countOfTeachers);

    return SUCCESS;
}

int ShowAllFromFile(int fdMain)
{
    unsigned int size = 0;
    int count = 1;
    struct department temp;
    int executionCode = 0;
```

```
    executionCode = lseek(fdMain, 0, SEEK_SET);
    if (executionCode == ERROR_CODE)
    {
        printf(ERROR_READ_FROM_FILE);
        return ERROR_CODE;
    }

    if (CheckRead(read(fdMain, &size, sizeof (unsigned int))) == ERROR_CODE)
        return ERROR_CODE;

    while(CheckRead(read(fdMain, &temp, size)))
    {
        printf("%d) Название: %s;  Количество профессоров: %g;  Количество
преподавателей: %g.\n",
               count, temp.name, temp.countOfProfessors, temp.countOfTeachers);

        count++;
    }

    return SUCCESS;
}

int DeleteElementFromFile(char** argvMain, int fdMain)
{
    if (argvMain[FIRST_PARAM] == NULL)
    {
        printf(ERROR_INPUT_IS_EMPTY);
        return ERROR_CODE;
    }

    int number = 0;

    if (atoi(argvMain[FIRST_PARAM]) != InputChecker(argvMain[FIRST_PARAM]))
    {
        printf(ERROR_INCORRECT_NUMBER_OF_ELEMENT);
        return ERROR_CODE;
    }
    else
        number = InputChecker(argvMain[FIRST_PARAM]);

    int size = 0;

    lseek(fdMain, 0, SEEK_SET);

    if (read(fdMain, &size, sizeof (unsigned int)) == ERROR_CODE)
    {
        printf(ERROR_READ_FROM_FILE);
        return ERROR_CODE;
    }

    struct department temp;

    char* path = TEMP_FILE_NAME;
    mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH;
    int supFid = open(path, O_RDWR | O_CREAT, mode);

    int count = 0;
    DeleteAllFromFile(supFid);

    while(CheckRead(read(fdMain, &temp, size)))
```

```
    {
        if (write(supFid, &temp, size) == ERROR_CODE)
        {
            printf(ERROR_READ_FROM_FILE);
            close(supFid);
            return ERROR_CODE;
        }

        count++;
    }

    if (InputChecker(argvMain[FIRST_PARAM]) > count)
    {
        printf(ERROR_INCORRECT_NUMBER_OF_ELEMENT);
        close(supFid);
        return ERROR_CODE;
    }

    DeleteAllFromFile(fdMain);
    lseek(fdMain, 0, SEEK_SET);
    lseek(supFid, 0, SEEK_SET);

    if (write(fdMain, &size, sizeof (unsigned int)) == ERROR_CODE)
    {
        printf(ERROR_OUTPUT_IN_FILE);
        close(supFid);
        return ERROR_CODE;
    }

    int executionCode = 0;

    for (int i = 1; i <= count; i++)
    {
         executionCode = read(supFid, &temp, size);

        if (executionCode == ERROR_CODE || executionCode < sizeof (unsigned int))
        {
            printf(ERROR_READ_FROM_FILE);
            return ERROR_CODE;
        }

         if (i == number)
             continue;

        if (write(fdMain, &temp, size) == ERROR_CODE)
        {
            printf(ERROR_OUTPUT_IN_FILE);
            close(supFid);
            return ERROR_CODE;
        }
    }

    close(supFid);
    remove(TEMP_FILE_NAME);

    return SUCCESS;
}

int FirstRequest(int fdMain)
{
```

```
    int size = 0;
    int count = 0;
    struct department temp;

    if (lseek(fdMain, 0, SEEK_SET) == ERROR_CODE)
    {
        printf(ERROR_READ_FROM_FILE);
        return ERROR_CODE;
    }

    if (CheckRead(read(fdMain, &size, sizeof (unsigned int))) == ERROR_CODE)
    {
        printf(ERROR_READ_FROM_FILE);
        return ERROR_CODE;
    }

    while(CheckRead(read(fdMain, &temp, size)))
    {
        count++;

        if (temp.countOfProfessors == 0)
            printf("На кафедре %d - \"%s\" нет профессоров!\n",
               count, temp.name);
    }

    if (count == 0)
        printf(ERROR_DEPARTMENTS_IS_MISSING);

    return SUCCESS;
}

int SecondRequest(int fdMain)
{
    int size = 0;
    int count = 0;
    double per = 0;
    struct department temp;

    if (lseek(fdMain, 0, SEEK_SET) == ERROR_CODE)
    {
        printf(ERROR_READ_FROM_FILE);
        return ERROR_CODE;
    }

    if (CheckRead(read(fdMain, &size, sizeof (unsigned int))) == ERROR_CODE)
        return ERROR_CODE;

    while(CheckRead(read(fdMain, &temp, size)))
    {
        if (temp.countOfProfessors / temp.countOfTeachers > per)
            per = temp.countOfProfessors / temp.countOfTeachers;
    }


    if (lseek(fdMain, 0, SEEK_SET) == ERROR_CODE)
    {
        printf(ERROR_READ_FROM_FILE);
        return ERROR_CODE;
    }
```

```
    if (CheckRead(read(fdMain, &size, sizeof (unsigned int))) == ERROR_CODE)
    {
        printf(ERROR_READ_FROM_FILE);
        return ERROR_CODE;
    }

    if (per == INFINITY)
    {
        printf(ERROR_DEPARTMENTS_IS_MISSING);
        return SUCCESS;
    }

    printf(SECOND_REQUEST_MASSAGE);

    while(CheckRead(read(fdMain, &temp, size)))
    {
        count++;

        if (per == temp.countOfProfessors / temp.countOfTeachers)
            printf("%d) \"%s\" - %.2f%%\n", count, temp.name, per * 100);
    }

    return SUCCESS;
}
```

Листинг 5 – содержание файла makefile проекта SP_4:

```
CC = gcc
CFLAGS = -Wall -g -std=c99
SOURCES = main.c functions.c FileFunctions.c
OBJECTS = $(SOURCES:.c=.o)
OUTFILE = Departments

all: $(OUTFILE)
$(OUTFILE): $(OBJECTS)
  $(CC) $(OBJECTS) -o $@

%.o: %.c
  $(CC) -c $(CFLAGS) $< -o $@

.PHONY: clean
clean:
  rm $(OUTFILE) $(OBJECTS)
```

makefile

```
CC = gcc
CFLAGS = -Wall -g -std=c99
SOURCES = main.c functions.c FileFunctions.c
OBJECTS = $(SOURCES:.c=.o)
OUTFILE = Departments

all: $(OUTFILE)

$(OUTFILE): $(OBJECTS)
    $(CC) $(OBJECTS) -o $@

%.o: %.c
    $(CC) -c $(CFLAGS) $< -o $@

.PHONY: clean
clean:
    rm $(OUTFILE) $(OBJECTS)

```

Рисунок 1 – содержание файла makefile

1) Примеры работы:

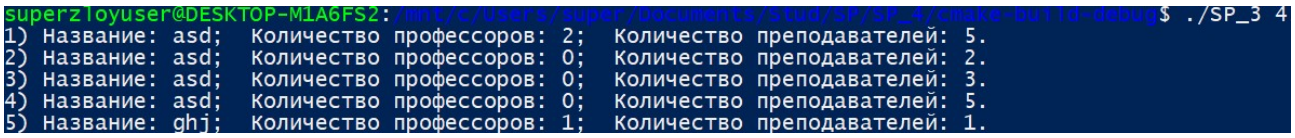

Рисунок 2 – результат работы с пунктом меню «4»

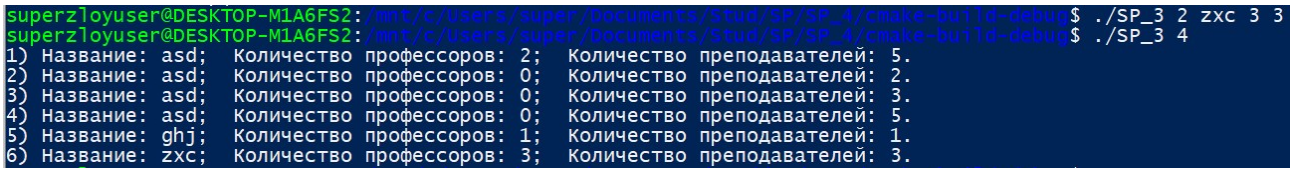

Рисунок 3 – результат работы с пунктом меню «2»

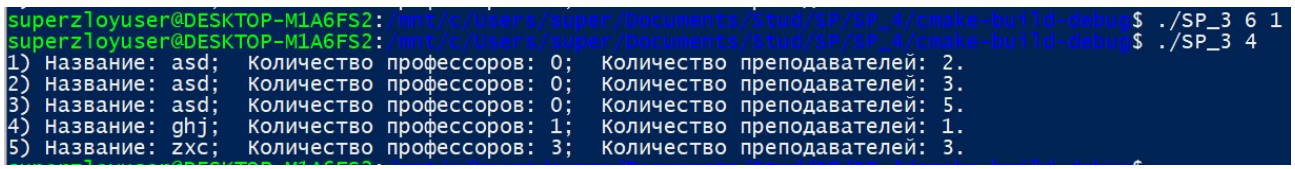

Рисунок 4 - результат работы с пунктом меню «6»

```
superzloyuser@DESKTOP-M1A6FS2:/mnt/c/Users/super/Documents/Stud/SP/SP_4/cmake-build-debug$ ./SP_3 8
На кафедре 1 - "asd" нет профессоров!
На кафедре 2 - "asd" нет профессоров!
На кафедре 3 - "asd" нет профессоров!
```

Рисунок 5 – результат работы с пунктом меню «8»

```
superzloyuser@DESKTOP-M1A6FS2:/mnt/c/Users/super/Documents/Stud/SP/SP_4/cmake-build-debug$ ./SP_3 9
Кафедры, на которых доля профессоров максимальна:
4) "ghj" - 100.00%
5) "zxc" - 100.00%
```

Рисунок 6 – результат работы с пунктом меню «9»

5 Вывод

В ходе данной лабораторной работы были изучены азы работы с файлами в программах для ОС GNU/Linux.