

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Программирование»
Тема: Обход файловой системы

Студент гр. 0382

Азаров М.С.

Преподаватель

Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Изучить основные принципы работы с файловой системой.

Задание.

Дана некоторая корневая директория, в которой может находиться некоторое количество папок, в том числе вложенных. В этих папках хранятся некоторые текстовые файлы, имеющие имя вида <filename>.txt. Требуется найти файл, который содержит строку "Minotaur" (файл-минотавр). Файл, с которого следует начинать поиск, всегда называется file.txt (но полный путь к нему неизвестен). Каждый текстовый файл, кроме искомого, может содержать в себе ссылку на название другого файла (эта ссылка не содержит пути к файлу). Таких ссылок может быть несколько.

Пример:

Содержимое файла **a1.txt**

```
@include a2.txt  
@include b5.txt  
@include a7.txt
```

А также файл может содержать тупик:

Содержимое файла **a2.txt**

```
Deadlock
```

Программа должна вывести правильную цепочку файлов (с путями), которая привела к поимке файла-минотавра.

Пример

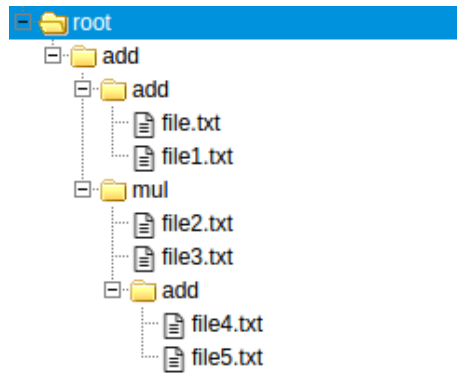


Рисунок 1: Пример файловой системы задания

file.txt:

```
@include file1.txt
@include file4.txt
@include file5.txt
```

file1.txt:

```
Deadlock
```

file2.txt:

```
@include file3.txt
```

file3.txt:

```
Minotaur
```

file4.txt:

```
@include file2.txt
@include file1.txt
```

file5.txt:

```
Deadlock
```

Правильный ответ:

```
./root/add/add/file.txt  
./root/add/mul/add/file4.txt  
./root/add/mul/file2.txt  
./root/add/mul/file3.txt
```

Цепочка, приводящая к файлу-минотавру может быть только одна.

Общее количество файлов в каталоге не может быть больше 3000.

Циклических зависимостей быть не может.

Файлы не могут иметь одинаковые имена.

Ваше решение должно находиться в директории **/home/box**, файл с решением должен называться **solution.c**. Результат работы программы должен быть записан в файл **result.txt**. Ваша программа должна обрабатывать директорию, которая называется **labyrinth**.

Основные теоретические положения.

Рассмотрим основные функции для работы с деревом файловой системы, объявления которых находятся в заголовочном файле **dirent.h** (также, может понадобится включить заголовочный файл **sys/types.h**)

Для того, чтобы получить доступ к содержимому некоторой директории можно использовать функцию

```
DIR *opendir(const char *dirname);
```

Которая возвращает указатель на объект типа **DIR** с помощью которого можно из программы работать с заданной директорией.

Тип **DIR** представляет собой поток содержимого директории. Для того, что бы получить очередной элемент этого потока, используется функция

```
struct dirent *readdir(DIR *dirp);
```

Она возвращает указатель на объект структуры **dirent**, в котором хранится информация о файле. Основным интерес представляют поля, хранящие имя и тип объекта в директории (это может быть не только "файл" и "папка").

После завершения работы с содержимым директории, необходимо вызвать функцию

```
int closedir(DIR *dirp);
```

Передав ей полученный функцией **readdir()** ранее дескриптор.

Рассмотрим пример функции, которая печатает содержимое заданной директории:

```
void list_dir(const char *dirPath)
{
    DIR *dir = opendir(dirPath);           // "открываем" директорию
    if(dir) {                               // если это удалось успешно
        struct dirent *de = readdir(dir);  // получаем очередной элемент открытой
директории
        while (de) {                        // если это удалось
            printf("%s/%s\n", dirPath, de->d_name);    // печатаем имя этого элемента
            de = readdir(dir);              // снова получаем очередной элемент открытой директории
        }
    }
    closedir(dir);                          // не забываем "закрыть" директорию
}
```

Выполнение работы.

Структуры:

- **labyrinth:**

Содержит в себе два массива *char** path* - пути к файлам, *char** f_name* — названия файлов, и *int n*- количество файлов, *int max_n* — текущее максимально допустимое количество файлов .

- **Stack :**

Структура данных в виде стека , содержит *char** arr* — массив данных, *int n* — количеств элементов, *int max_n* - текущее максимально допустимое количество элементов.

Функция *get_labyrinth()* :

Описание:

Функция ищет и сохраняет названия и путь всех найденных вложенных файлов в переданной начальной директории.

Переменные :

- Параметры функции:
 - *char* old_path* — начальная директория с которой идет поиск файлов.
 - *labyrinth* lab* — структура в которую сохраняются названия найденных файлов и пути к ним.
- Локальные переменные:
 - *DIR* dir* - указатель на структуру типа *DIR*, которая содержит информацию о открытом каталоге с помощью функции *opendir()*.
 - *char path[100]* — вспомогательная переменная для копирования изначального *old_path* и последующего его изменения.

- *int len* — тоже вспомогательная переменная для возврата измененного *path* к изначальному виду.
- *struct dirent* inf_file* — указатель на структуру, которая содержит информацию о файле в открытой директории.

Ход работы :

- Открываем директорию по пути *old_path*.
- Копируем *old_path* в *path*.
- Проверяем успешно ли открылась директория.
- Получаем информацию о первом файле в директории в переменную *inf_file*.
- Пока есть файлы в директории :
 - Если файл это папка, то снова вызываем *get_labyrinth()* для поиска файлов но уже в найденной директории.
 - Если файл это обычный файл, то сохраняем в структуру *lab* название найденного файла и путь к нему.
 - И получаем информацию о следующем файле в этой директории.
- Закрываем открытую директорию.

Функция *found_minotaur()*:

Описание :

Ищет файл-минотавр среди файлов *lab*.

Переменные:

- Параметры функции:

- *labyrinth lab* - структура в которую сохраняются названия найденных файлов и пути к ним.
- *char* f_name* — имя файла с которого нужно начинать искать
- *Stack* ans* — структура для сохранения ответа в формате стек.
- Локальные переменные
 - *char str[50]* - стока для чтения файла
 - *char f_name_next[50]* - вспомогательная строка для удаления из считанной строки "@include "
 - *char* t* - вспомогательный указатель для поиска и удаления «\n» в считанной строке
 - *char tmp[100];* - вспомогательная строка для конкатенации строк

Ход работы :

- Среди всех найденных файлов в *lab* ищем файл с именем *f_name*.
- После нахождения файла открываем найденный файл для чтения по соответствующему пути хранящейся *lab*.
- Читаем первую строку в файле и удаляем «\n», если он есть.
- Если считанная строка "Deadlock" , то закрываем файл и выходим из текущей функции , возвращая 0.
- Если считанная строка "Minotaur", то закрываем файл добавляем в *ans* путь к текущему файлу возвращаем 1.
- Если ничего выше не подошло, значит остался только вариант ссылок на другие файлы. Пока есть строки в файле :
 - Удаляем из считанной строки *str* "@include " и если есть «\n».
 - По имени файла , полученный из строки *str* , ищем и читаем новый файл с помощью *found_minotaur()*.

- Если функция *found_minotaur()* вернула 1 то значит файл-минотавр был найден. Поэтому закрываем файл, добавляем в *ans* путь к текущему файлу , возвращаем 1.
- Если среди ссылок не было пути к файлу-минотавр , то закрываем файл возвращаем 0.

Функция *push(Stack* ans):*

Описание :

Возвращает последнюю добавленную строку в *Stack*.

Функция *pop(Stack* ans, char* path):*

Описание :

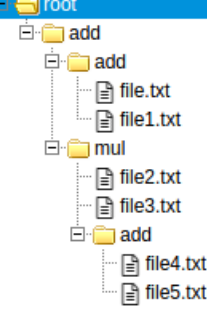
Добавляет переданную строку в переданную структуру *Stack*.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
-------	----------------	-----------------	-------------

<p>1.</p>	 <pre> graph TD root[root] --> add1[add] root --> mul[mul] add1 --> file1[file1.txt] add1 --> file2[file2.txt] mul --> file3[file3.txt] mul --> add2[add] add2 --> file4[file4.txt] add2 --> file5[file5.txt] </pre> <p><u>file.txt:</u></p> <pre>@include file1.txt @include file4.txt @include file5.txt</pre> <p><u>file1.txt:</u></p> <pre>Deadlock</pre> <p><u>file2.txt:</u></p> <pre>@include file3.txt</pre> <p><u>file3.txt:</u></p> <pre>Minotaur</pre> <p><u>file4.txt:</u></p> <pre>@include file2.txt @include file1.txt</pre> <p><u>file5.txt:</u></p> <pre>Deadlock</pre>	<pre>./root/add/add/file.txt ./root/add/mul/add/ file4.txt ./root/add/mul/file2.txt ./root/add/mul/file3.txt</pre>	<p>Программа работает правильно</p>
-----------	---	--	---

Выводы.

Была изучены основные принципы работы с файловой системой.

Разработана программа, выполняющая поставленную задачу, а именно поиск «файла-минотавр» в «файловом лабиринте» . Для решения этой задачи были использованы полученные знания о том как устроена файловая система и как с ней можно взаимодействовать .

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb3_2.c

```
#include <stdio.h>
#include <dirent.h>
#include <stdlib.h>
#include <string.h>

#define START_DIR  "./labyrinth"
#define RESULT_FILE  "./result.txt"
#define START_FILE  "file.txt"
#define MAX_LEN_PATH 100
#define MAX_LEN_F_NAME 15
#define INIT_SIZE_LAB 50
#define INIT_SIZE_ANS 50

struct labyrinth {
    char** path; //путь к файлу
    char** f_name; //имя файла
    int n; // количество файлов
    int max_n; //текущее максимальное количество файлов
};

typedef struct labyrinth labyrinth;

struct Stack {
    char** arr;
    int n;
    int max_n;
};

typedef struct Stack Stack;
```

```

char* my_strcat(char* s12 , char* s1, char* s2) {

    strcpy(s12, s1);
    strcat(s12, s2);
    return s12;
}

void get_labyrinth(char* old_path, labyrinth* lab) { //Опис.:
рекурсивно находит все файлы и пути к ним
    DIR* dir = opendir(old_path); //открываем директорию по пути
old_path
    char path[100];
    int len;

    strcpy(path, old_path); //копируем old_path в path
    strcat(path, "/");

    if (dir == NULL) { //проверяем успешно ли открыта дир.
        printf("ERROR: dir = NULL");
        return;
    }

    struct dirent* inf_file = readdir(dir); // получаем инф о первом
файле в дир.
    while (inf_file) { //пока есть файлы в дир.

        if ( (inf_file->d_type == DT_DIR) && (strcmp(inf_file->d_name,
".")) != 0 //если это папка
            && (strcmp(inf_file->d_name, "..")) != 0 ) {

            len = strlen(path);
            get_labyrinth(strcat(path, inf_file->d_name), lab);
//рекурсия: снова вызываем get_labyrinth() для поиска
            path[len] = '\0'; //
файлов но уже в найденной дир.

```

```

    }

    if (inf_file->d_type == DT_REG) { //если это обычный файл
        if (lab->n == lab->max_n - 1) { //проверяем есть ли в lab
место для нового файла
            lab->max_n += 50;
            lab->f_name = realloc(lab->f_name, lab->max_n*sizeof
(char*));
            lab->path = realloc(lab->path, lab->max_n*sizeof
(char*));
        }

        char* tmp_f_name = malloc(MAX_LEN_F_NAME * sizeof (char));
//
        char* tmp_path = malloc(MAX_LEN_PATH * sizeof (char));

        strcpy(tmp_f_name , inf_file->d_name);
        strcpy(tmp_path, path);

        lab->f_name[lab->n] = tmp_f_name; //сохраняем в структуру
lab название найденного файла
        lab->path[lab->n] = tmp_path; //сохраняем в структуру lab
путь к найденному файлу
        lab->n ++;

    }

    inf_file = readdir(dir); // получаем инф о след. файле в дир.
}

closedir(dir); //закрываем дир.
return;
}

```

```

void pop(Stack* ans, char* path) {
    char *tmp = malloc(MAX_LEN_PATH * sizeof (char));
    if (ans->n == ans->max_n - 1) { //пров-ка на наличии свобод. места
для доб.
        ans->max_n += 50;
        ans->arr = realloc(ans->arr, ans->max_n*sizeof (char*));
    }

    strcpy(tmp, path);

    ans->arr[ans->n] = tmp;
    ans->n++;

    return;
}

char* push(Stack* ans) {
    if ( ans->n == 0) {
        return NULL;
    }

    ans->n --;
    return  ans->arr[ans->n];
}

int found_minotaur(labyrinth lab, char* f_name, Stack* ans) {
//рекурсивно ищет минотавра
    int i;
    char str[50], f_name_next[50]; //стока для чтения файла ,
вспомогательная строка для удаления "@include "
    char* t; // вспомогательный указателя для поиска и удаления \n
в считанной строке
    char tmp[100]; //вспомогательная строка для конкатенации строк

    for (i = 0; i < lab.n; i++) { //среди всех найденных файлов в lab

```

```

        if (strcmp(lab.f_name[i], f_name) == 0) {    //ищет файл с
именем f_name

                //после нахождения файла
                FILE* f = fopen( my_strcat(tmp, lab.path[i],
lab.f_name[i]) ,"r");  //открываем найденный файл для чтения по

// соответствующему пути  в lab
                fgets(str, 50, f); // читаем первую строку в файле
                t = strchr(str, '\n');          //удаляем \n, если он есть
                if ( t != NULL) {
                        *t = '\0';
                }

                if (strcmp(str, "Deadlock") == 0) { //если считанная
строка "Deadlock"

                        fclose(f); //закрываем файл
                        return 0;   //и выходим из текущей фун , возвращая 0
                }

                if (strcmp(str, "Minotaur") == 0) { //если считанная
строка  "Minotaur"

                        fclose(f); //закрываем файл
                        pop(ans, my_strcat(tmp, lab.path[i], lab.f_name[i]));
//добавляем в ans путь к текущему файлу
                        return 1; //возвращаем 1
                }

do { //если ничего выше не подошло значит остался только
        // вариант  ссылок на другие файлы
        strcpy(f_name_next , str+9); //удаляем "@include "
        t = strchr(f_name_next, '\n'); //удаляем \n, если он
есть

                if ( t != NULL) {
                        *t = '\0';
                }

```



```

        if ( found_minotaur(lab, f_name_next, ans) == 1) { //
ищем и читаем новый файл найденый в тукущем файле
                                                                    //
с помощью found_minotaur()
                                                                    //

если фун вернула 1 то значит фай-минотавр был найден
        fclose(f); //закрываем файл
        pop(ans, my_strcat(tmp, lab.path[i],
lab.f_name[i])); //добавляем в ans путь к текущему файлу
        return 1; //возвращаем 1

    }
    } while (fgets(str, 50, f)); //читаем файл пока в нем есть
строки

        fclose(f); //если среди ссылок небыло путя к файлу-
минотавр , то закрываем файл
        return 0; //возвращаем 0

    }
}

void write_answer(Stack* ans){
    char* tmp;
    FILE* f = fopen(RESULT_FILE, "w");

    tmp = push(ans);
    while (tmp){
        fprintf(f, "%s\n", tmp);
        tmp = push(ans);
    }

    fclose(f);

    return;
}

```

```

void clear_dynamic_arr(labyrinth* lab, Stack* ans) {
    int i;

    for (i = 0; i < lab->n ; i++){
        free(lab->f_name[i]);
        free(lab->path[i]);
    }
    free(lab->f_name);
    free(lab->path);

    for (i = 0; i < ans->n ; i++) {
        free(ans->arr[i]);
    }
    free(ans->arr);
}

int main() {
    labyrinth lab = {.n = 0, .max_n = INIT_SIZE_LAB};
    Stack answer = {.n = 0, .max_n = INIT_SIZE_ANS};

    answer.arr = malloc(answer.max_n * sizeof (char*));

    lab.f_name = malloc(lab.max_n * sizeof (char*));
    lab.path = malloc(lab.max_n * sizeof (char*));

    get_labyrinth(START_DIR, &lab);
    found_minotaur(lab, START_FILE, &answer);

    write_answer(&answer);

    //незабудь очистить ans и lab!!!

```

```
//незабыл)

clear_dynamic_arr(&lab, &answer);

return 0;
}
```