МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Математического обеспечения электронно-вычислительных машин

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Программирование»

ТЕМА: ОБХОД ФАЙЛОВОЙ СИСТЕМЫ

ВАРИАНТ: 1

Студентка гр. 0382 _____ Рубежова Н.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Изучить обход файловой системы, его реализацию в языке Си, освоить функции для работы с деревом файловой системы, используя их в программном коде.

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.

Основные теоретические положения.

Рассмотрим основные функции для работы с деревом файловой системы, объявления которых находятся в заголовочном файле *dirent.h* (также, может понадобиться 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, чтобы 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. Основной 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.

Выполнение работы.

1. Опишем структуру *File*, которая будет хранить сведения о файле: имя файла и путь к нему. Чтобы каждый раз не писать *struct File* воспользуемся оператором *typedef* и переопределим тип *struct File* как new*File*. У структуры new*File* будет 2 поля: *char* fname, char* fpath*, которые будут хранить строки – название файла и путь к нему соответственно.

2. Опишем структуру *struct rewersed_arr*, которая будет хранить массив строчек ответа-вывода, то есть цепочку путей до файла минотавра, но

в обратном порядке, так как с помощью функции *push_answer()* пути добавляются во время выполнения рекурсивной(!!!) функции *findMntr()*.Чтобы каждый раз не писать *struct rewersed_arr* воспользуемся оператором *typedef* и переопределим тип *struct rewersed_arr* как *Paths_arr*. У структуры *Paths_arr* будет 3 поля: *char** arr, int count, int maxcount*, которые будут хранить массив-цепочку путей, текущее количество элементов в массиве и число элементов, под которое выделена память(именно оно будет меняться перед выполнением функции *realloc())*, соответственно.

3. Определим функцию *void setName()*, которая записывает имя файла – строчку *fname*, переданную в качестве аргумента функции - в поле *fname* для файла, переданного указателем на структуру *File* ptrFile*. Обращаемся к полю структуры через указатель: *ptrFile->fname*, и выделяем память под строку-имя файла с помощью *calloc()*. Далее просто копируем переданную строку в поле с помощью функции *strcpy()*.

4. Определим функцию *setPath()*, которая находит путь к файлу с определенным именем. То есть функции передается *const char* pathDir* - путь к директории, в которой будет осуществляться поиск, *newFile* ptrFile* – указатель на структуру файла, в которой уже заранее заполнено поле *fname* – имя файла, который предстоит найти. Задача функции – заполнить у переданной структуры поле *fpath* – путь к искомому файлу. Для обработки директории нам понадобится дескриптор директории *DIR* dir*, возвращаемый функцией *opendir(pathDir)*. А чтобы поочередно «выбирать» файлы из директории воспользуемся функцией *readdir(dir)*, которая возвращает тип *struct dirent** с удобными полями: *d_type, d_name*.

5. Определим функцию *push_answer()*, которая добавляет элемент-путь к файлу, нужному для поимки файла-минотавра, в массив структуры *Paths_arr*, а также расширяет память, если ее недостаточно для добавления очередного элемента, и выделяет память под непосредственно саму строку-путь. Аргументами функции являются: указатель на структуру массива ответа-

вывода *Paths_arr* ptrAnswer* и сама строка-путь *char* path*, которую нужно добавить в массив в качестве элемента.

6. Определим одну из основных функций этой программы *findMntr()*. Она рекурсивна. Ей передаются путь к корневой директории *const char* root*, имя файла, с которого нужно начинать поиск минотавра, *char* fname* и указатель *Paths_arr* ptrAnswer* на структуру массива, в который будет записываться цепочка путей, ведущих к поимке минотавра.

Функция возвращает *int* значение: 1 – если через этот файл можно дойти до минотавра, 0 – если этот путь отсутствует в цепочке файлов для поимки минотавра, эти значения нам понадобятся для регулировки рекурсии.

Опишем алгоритм работы функции. Функция создаёт элемент-структуру *newFile file,* затем вызывает для нее функцию *setName()* и *setPath()*. Таким образом получаем структуру *file* с заполненными полями *fname* и *fpath*. Далее мы можем уже работать с ним. Откроем его на чтение содержимого с помощью функции *fopen()*, а также не забудем закрывать его с помощью *fclose()* перед завершением выполнения функции. Считываем первую строчку файла с помощью *fscanf()*, чтобы проверить, к какому типу файл отнести: файл-минотавр, файл-тупик или файл с ссылками на названия других файлов. Если это файл-минотавр, то добавляем его в *Path_arr*, поэтому вызываем *push_answer(),* закрываем файл и возвращаем значение 1. Если это файл-тупик, закрываем файл и возвращаем 0. Если это файл со ссылками, то мы должны считать названия файлов, на которые ссылаются. Мы снова должны считать содержимое файла, но так как первый *fscanf()* уже был, нам нужно переместить указатель в потоке на начало, чтобы не упустить первую строку с названием. Переместим указатель в потоке на начало с помощью функции *fseek()*. Чтобы считывать несколько строчек запустим цикл *while(fscanf(fFile,"%s%s",tmp,s)!=EOF)*. В *s* будет считываться название файла, поэтому мы можем передавать его дальше в рекурсивную функцию в качестве ключа *key*, чтобы проверить, входит ли файл, который среди ссылок, в цепочку путей, по которым можно достичь минотавра. Поэтому проверяем значение

рекурсивной функции *if(findMntr(root,key,ptrAnswer)==1),* т.е. если через файл-ссылку можно каким-то образом дойти до минотавра, то и наш файл, который на него ссылается, участвует в этой цепочке. Поэтому добавляем его в *Paths_arr* с помощью *push_answer(),* закрываем файл и возвращаем значение 1, так как этот файл участвует в цепочке файлов.

7. Определим функцию *main().* Установим путь обрабатываемой директории и файл, с которого следует начать поиск минотавра согласно условию: *const char* root="./labyrinth";* и *char* fname="file.txt";*

Объявим структуру массива ответа-вывода *Paths_arr answer,* чтобы потом в функции передавать указатель на эту структуру и записывать туда ответ. Определим, обращаясь к полям структуры, число элементов массива *answer.maxcount=10*, под которое пока что следует выделить память и потом, в случае чего, ее расширять, текущее число элементов в массиве, равное 0, *answer.count=0.* И выделим небольшое количество памяти под этот массив с помощью функции *calloc(),* напомню, что случай расширения памяти мы учли в функции *push_answer().*

Вызываем функцию с установленными начальными данными *findMntr(root,fname,&answer);* По результату у нас будет заполненный массив *Paths_arr,* однако пути цепочки будут записаны в обратном порядке, так как функция *findMntr()* рекурсивна. Поэтому откроем файл на запись результата: FI*LE* fRes=fopen("result.txt","w");*

Запустим цикл *for* и, перебирая элементы-пути массива структуры *Paths_arr* в обратном порядке, будем записывать строчки-пути в файл, открытый на запись, а затем освобождать память, выделенную под эту строчку в массиве. После прохождения цикла, вся цепочка будет записана, память под строчки-пути в массиве освобождена. Осталось освободить память, выделенную под сам массив, и закрыть файл, открытый на запись результата. Таким образом, по завершению программы цепочка файлов, приводящих к поимке минотавра, будет записана в файл *result.txt.*

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Входных данных нет, обрабатывается директория из примера, данного в описании лабораторной работы.	Вывод, записанный в *result.txt:* *./labyrinth/add/add/file.txt* *./labyrinth/add/mul/add/file4.txt* *./labyrinth/add/mul/file2.txt* *./labyrinth/add/mul/file3.txt*	Вывод верный
2.	Входных данных нет, обрабатывается директория *labyrinth*	Вывод записан в *result.txt,* проверяется системой *moodle*	Вывод верный

Выводы.

Был изучен обход файловой системы, его реализация в языке Си, а также освоены функции для работы с деревом файловой системы посредством использования их в программном коде.

Разработана программа, которая обходит некоторую корневую директорию с вложенными папками и файлами, находит так называемый «файл-минотавр» и выводит правильную цепочку файлов(с путями), которая привела к его поимке.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <dirent.h>

typedef struct File{
    char* fname;
    char* fpath;
}newFile;

typedef struct rewersed_arr{
    char** arr;
    int count;
    int maxcount;
}Paths_arr;

void setName(newFile* ptrFile,const char* fname){
    ptrFile->fname=calloc(50,sizeof(char));
    strcpy(ptrFile->fname,fname);
}

void setPath(const char* pathDir,newFile* ptrFile){
    char nextDir[300]={0};
    strcpy(nextDir,pathDir);
    strcat(nextDir,"/");
    DIR* dir=opendir(pathDir);
    if(!dir)
        return;
    struct dirent* de=readdir(dir);
    while(de){

if(de->d_type==DT_DIR&&strcmp(de->d_name,".")!=0&&strcmp(de->d_name,".
.")!=0){
                int len=strlen(nextDir);
                strcat(nextDir,de->d_name);
                setPath(nextDir,ptrFile);
                nextDir[len]='\0';
            }

if(de->d_type==DT_REG&&strcmp(de->d_name,ptrFile->fname)==0){
                strcat(nextDir,de->d_name);
                ptrFile->fpath=calloc(300,sizeof(char));
                strcpy(ptrFile->fpath,nextDir);
                closedir(dir);
```

```
                return;
            }
            de=readdir(dir);
        }
    }

    void push_answer(Paths_arr* ptrAnswer, char* path){
        if(ptrAnswer->count==ptrAnswer->maxcount){
            ptrAnswer->maxcount+=5;

ptrAnswer->arr=realloc(ptrAnswer,ptrAnswer->maxcount*sizeof(char*));
        }

ptrAnswer->arr[ptrAnswer->count]=calloc(strlen(path)+2,sizeof(char));
        strcpy(ptrAnswer->arr[ptrAnswer->count++],path);
    }

    int  findMntr(const  char*  root,const  char*  fname,Paths_arr*
ptrAnswer){
        newFile file;
        setName(&file,fname);
        setPath(root,&file);
        FILE* fFile=fopen(file.fpath,"r");
        if(!fFile)
            return;
        char s[50];
        fscanf(fFile,"%s",s);
        if(strcmp(s, "Minotaur") == 0){
            push_answer(ptrAnswer,file.fpath);
            fclose(fFile);
            return 1;
        }
        else if(strcmp(s,"Deadlock")==0) {
            fclose(fFile);
            return 0;
        }
        else {
            char tmp[50],key[50];
            fseek(fFile,0,SEEK_SET);
            while(fscanf(fFile,"%s%s",tmp,s)!=EOF){
                strcpy(key,s);
                if(findMntr(root,key,ptrAnswer)==1){
                    push_answer(ptrAnswer,file.fpath);
                    fclose(fFile);
                    return 1;
                }
            }
        }
    }
```

```
int main(){
    const char* root="./labyrinth";
    char* fname="file.txt";

    Paths_arr answer;
    answer.maxcount=10;
    answer.count=0;
    answer.arr=calloc(answer.maxcount,sizeof(char*));

    findMntr(root,fname,&answer);

    FILE* fRes=fopen("result.txt","w");
    int i;
    for(i=answer.count-1;i>=0;i--){
        fprintf(fRes,"%s\n",answer.arr[i]);
        free(answer.arr[i]);
    }
    free(answer.arr);
    fclose(fRes);
    return 0;
}
```