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. 1304 _____ Арчибасов Е.О.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2022

Цель работы.

Изучение принципов работы с деректориями на ОС Linux на языке Си.

Задание.

Вариант 1

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 .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.

Рекурсия в программировании — это вызов функции (или же процедуры) непосредственно из самой себя. Есть простая (непосредственная) рекурсия или рекурсия, которая работает через другие процедуры и функции (такой вид называется косвенной, сложной рекурсией). Количество вложенных вызовов функции или процедуры называется глубиной рекурсии. Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причём без явных повторений частей программы и использования циклов.

Выполнение работы.

В функции main() определяется начальное положение для поиска слова, создаются массивы символов для записи текущей деректории/файла и пути. Далее вызывается функция *takePath()*, которая пишет текущий к текущей деректории. После следует функция *minotavr() – функция*, проверяющая дальнейшие пути к файлу. Если на данном шаге не достигнут результат или тупик, функция вызывается повторно. В это время в функции *takePath()* пишется путь к текущей успешной папке – все это выполняется рекурсивно. Помимо стандартной библиотеки языка, были использованы dirent.h и sys/stat.h.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии

1.	Test	`./root/add/add/file.txt`
		`./root/add/mul/add/file4.txt`
		`./root/add/mul/file2.txt`
		`./root/add/mul/file3.txt`
			Программа работает корректно

Выводы.

В ходе выполнения работы 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.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

int takePath(char *labirint, char** names, char** ways, int* count){
   char tway[PATH_MAX + 1];
   DIR *dir = NULL;
   struct dirent *indir = NULL;
   dir = opendir(labirint);
   if(dir==NULL){ return -1;}
   indir=readdir(dir);
   while(indir != NULL){
      struct stat inf;
      if((strncmp( indir->d_name, ".", PATH_MAX) == 0) || (strncmp( indir-
>d_name, "..", PATH_MAX) == 0)){
         indir = readdir(dir);
         continue;
      }
      strncpy(tway, labirint, PATH_MAX);
      strncat(tway, "/", PATH_MAX);
      strncat(tway, indir->d_name, PATH_MAX);
      if(lstat(tway, &inf)== 0){
         if(S_ISDIR(inf.st_mode)){
            takePath(tway, names, ways, count);
         }
         else
         if(S_ISREG(inf.st_mode)){
         names[*count] = malloc(300*sizeof(char));
         (names[*count])[0]='\0';
         ways[*count] = malloc(300*sizeof(char));
         (ways[*count])[0]='\0';

            strcat(names[*count], indir->d_name);
            strcat(ways[*count], labirint);

            (*count)+=1;
         }
      }
```

```
        indir=readdir( dir );
    }

    closedir(dir);
    return 0;
}

int minotavr(char* name, char** names, char** ways, int count, char**
result, int* _count){
    int i;
    for(i=0; i<count;i++)
        if (strcmp(names[i], name)==0)
            break;
    char* way[300];
    way[0]='\0';
    strcat(way, ways[i]);
    strcat(way, "/");
    strcat(way, names[i]);

    FILE * file = fopen(way, "r");
    char str[300];
    str[0]='\0';
    char fname[300];
    char a[]="Minotaur";
    char b[]="Deadlock";
    char c[]="@include";
    fname[0]='\0';

    while(fscanf(file, "%s", str) != EOF){
        if(strcmp(str, c)==0){
            fscanf(file, "%s\n", fname);
            if (minotavr(fname, names, ways, count, result, _count)){
                result[*_count] = malloc(300*sizeof(char));
                (result[*_count])[0] = '\0';
                strcat(result[*_count], way);
                (*_count)++;
                fclose (file);
                return 1;
            }
        }
        else
        if(strcmp(str, a)==0){
            result[*_count] = malloc(300*sizeof(char));
            (result[*_count])[0] = '\0';
            strcat(result[*_count], way);
```

```
            (*_count)++;
            fclose (file);
            return 1;
        }
        else
        if(strcmp(str, b)==0){
            fclose (file);
            return 0;
        }
    }
    fclose (file);
    return 0;
}

int main(){
    char* names[5000];
    char* ways[5000];
    char* result[5000];
    char root[]="./labyrinth";
    char res[]="result.txt";
    int count=0;
    int tcount = 0;

    if (takePath( root, names, ways, &count) == 0)
        minotavr("file.txt", names, ways, count, result, &tcount);
    FILE * file = fopen(res, "w");
    while(tcount!=0){
        fputs(result[tcount-1], file);
        fputs("\n", file);
        printf("%s\n",result[tcount-1]);
        tcount--;
    }
    fclose (file);
    return 0;
}
```