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. 9303 _____ Максимов Е.А.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2020

Цель работы.

Изучить основные функции для работы с файловой системой.

Задание.

Вариант лабораторной работы №1.

Дана корневая директория *labyrinth*, в которой может находиться некоторое количество папок, в этих папках хранятся некоторые текстовые файлы, имеющие имя вида *<filename>.txt* или вложенные папки.

Требуется найти файл, который содержит строку "*Minotaur*".

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 (пример: *@include a2.txt*). Таких ссылок может быть несколько.

Основные теоретические положения.

Заголовочный файл *dirent.h* содержит необходимые функции для работы с файловой системой (директориями, файлами).

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

Данная функция используется для получения содержимого директории по пути *dirname*. Функция возвращает указатель на объект типа DIR с помощью которого можно из программы работать с заданной директорией (открытие директории).

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

Возвращает указатель на объект структуры dirent, в котором хранится информация о файле. Если файлы закончились, то возвращается нулевое значение.

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

Функция завершает работу с данной директорией (закрытие директории).

Выполнение работы.

В программе использовались следующие структуры:

1. Структура *Filetree*, состоящая из следующих полей:
    a. *char* name* – строка, содержащая название файла;
    b. *char* path* – строка, содержащая путь к файлу;
    c. *char* tailname* – строка, содержащая название файла, на которое ссылается данный файл; если файл содержит строку «Minotaur» или «Deadlock», то оно заменяет название этого файла.

В программе реализованы следующие функции:

1. Функция *Filetree* createLink(char* name, char* path, char* tailname)* принимает на вход 3 указателя на строки (название файла, путь к файлу и название ссылаемого файла. Функция динамически выделяет память для структуры и возвращает указатель на структуру.
2. Функция *void createLinkList(const char *name, int* count, Filetree*** list)* принимает на вход 3 указателя (указатель на путь к текущей директории, на количество обработанных файлов и указателей на массив указатель на структуры). Функция рекурсивно находит все файлы и записывает их как структуры в динамический массив указатель на структуры. Функция не имеет возвращаемого значения.
3. Функция *char** getPathList(int count, Filetree** list, int* pathListLength)* принимает на вход 3 указателя (количество обработанных файлов, указатель на массив указателей на структуры и указатель на количество результирующих строк). Функция находит структуру, которая связана с текстовым файлом, в котором содержится строка «Minotaur» и ищет структуру, в названии ссылки которой имеется название текущей структуры. Далее функция ищет структуру, которая связана с текстовым файлом, который содержит ссылку на последний обработанный файл. Это продолжается до тех пор, пока новая структура не обнаружится. Все пути файлов, с которыми связаны данные структуры динамически записываются в массив. Строки в массиве расположены в порядке, обратном необходимому. Функция возвращает указатель на массив строк.

4. Функция *void importPathList(char** pathList, int* pathListLength)* принимает на вход 2 указателя (массив строк с необходимыми путями к файлам и длину массива). Функция создаёт текстовый документ *result.txt,* записывает в него пути к файлам в необходимом порядке и освобождает динамическую память, выделенную для хранения данного массива строк. Функция не имеет возвращаемого значения.
5. Функция *void removeLinkList(int count, Filetree** list)* принимает на вход 2 значения (количество структур и указатель на массив структур). Функция освобождает динамическую память, выделенную для хранения данного массива структур (в том числе для полей структуры). Функция не имеет возвращаемого значения.

Разработанный программный код см. в приложении А.

Результаты тестирования.

Программный код был протестирован на онлайн-платформе «Stepik».

Результаты тестирования: 1 of 1 test(s) passed.

Выводы.

В ходе лабораторной работы были изучены функции для работы с директориями и файловой системой.

Была разработана программа, которая рекурсивно считывает файлы в исходной директории и находит цепочку файлов, с помощью которых можно найти текстовый файл с требуемым содержимым.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

#define UNIT_SIZE 512
#define START_DIRECTORY "./labyrinth"
#define BAD_END "Deadlock"
#define GOOD_END "Minotaur"
#define RESULT_FILE "./result.txt"

typedef struct Filetree{
      char* name;
      char* path;
      char* tailname;
}Filetree;

Filetree* createLink(char* name, char* path, char* tailname){
      Filetree* link = malloc(sizeof(Filetree));
      char* linkname = malloc(UNIT_SIZE*sizeof(char));
      link->name = strcpy(linkname, name);
      char* linkpath = malloc(UNIT_SIZE*sizeof(char));
      link->path = strcpy(linkpath, path);
      char* linktailname = malloc(UNIT_SIZE*sizeof(char));
      link->tailname = strcpy(linktailname, tailname);
      return link;
}

void createLinkList(const char *name, int* count, Filetree*** list){
    DIR *dir;
    struct dirent *element;
    if (!(dir = opendir(name))) return;
    while ((element = readdir(dir)) != NULL){
           char path[UNIT_SIZE];
        snprintf(path, sizeof(path), "%s/%s", name, element->d_name);
        if ((element->d_type == DT_DIR)&&(strcmp(element->d_name,
".."))&&(strcmp(element->d_name, ".."))) createLinkList(path, count, list);
else {
            FILE *fp = fopen(path, "r");
                char tailname[UNIT_SIZE];
                while(fgets(tailname, UNIT_SIZE, fp)!=NULL){
                      strtok(tailname, "\n");
                      (*count)++;
                      *list = realloc(*list, (*count)*sizeof(Filetree*));
                      if(strcmp(tailname, BAD_END)==0) (*list)[(*count)-1]
= createLink(element->d_name, path, BAD_END);
                      else if(strcmp(tailname, GOOD_END)==0)
(*list)[(*count)-1] = createLink(element->d_name, path, GOOD_END);
                      else (*list)[(*count)-1] = createLink(element-
>d_name, path, tailname+9);
                }
                fclose(fp);
```

```
        }
    }
    closedir(dir);
     return;
}

char** getPathList(int count, Filetree** list, int* pathListLength){
     int i=0;
     char** pathList = malloc((*pathListLength)*sizeof(char*));
     char* tempname = GOOD_END;
     while(1){
          for(i=0; i<count; i++) if(strcmp((list[i])->tailname,
tempname)==0) break;
          if(i==count) return pathList; else {
               (*pathListLength)++;
               pathList = realloc(pathList,
(*pathListLength)*sizeof(char*));
               pathList[(*pathListLength)-1] = list[i]->path;
               tempname = list[i]->name;
          }
     }
}

void importPathList(char** pathList, int* pathListLength){
     FILE* fp = fopen(RESULT_FILE, "w");
     for(int i=(*pathListLength)-1; i>=0; i--){
          fputs(pathList[i], fp);
          putc('\n', fp);
     }
     fclose(fp);
     free(pathList);
     return;
}

void removeLinkList(int count, Filetree** list){
     for(int i=0; i<count; i++){
          free(list[i]->name);
          free(list[i]->path);
          free(list[i]->tailname);
          free(list[i]);
     }
     free(list);
     return;
}

int main(){
     int count = 0;
     Filetree** list = malloc(sizeof(Filetree*)*count);
    createLinkList(START_DIRECTORY, &count, &list);
     int pathListLength = 0;
     char** pathList = getPathList(count, list, &pathListLength);
     importPathList(pathList, &pathListLength);
     removeLinkList(count, list);
    return 0;
}
```