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

ОТЧЕТ

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

Студент гр. 3343 _____ Гребнев Е.Д.

Преподаватель _____ Государкин Я.С.

Санкт-Петербург
2024

Цель работы

Создать программу на языке C, которая рекурсивно ищет определенные файлы в директориях, используя знания о работе с файлами и директориями.

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 .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.

Выполнение работы

Реализованный код на языке C выполняет поиск файла file.txt в директории "labyrinth" и ее поддиректориях. Если файл найден, программа открывает его и ищет строку, соответствующую FINAL_TARGET ("Minotaur"). Если такая строка найдена, программа заканчивает выполнение. Если в файле встречается строка, соответствующая DEADLOCK_TARGET ("Deadlock"), выполнение также завершается. В противном случае, программа ищет строки вида @include <file_name>, где <file_name> - это имя файла, и рекурсивно вызывает саму себя для поиска файла <file_name>. Поиск осуществляется в директории "labyrinth" и ее поддиректориях.

Когда все цели найдены, программа записывает пути к найденным файлам в обратном порядке в файл "result.txt".

Выводы

В процессе выполнения лабораторной работы был изучен синтаксис языка C для работы с файлами и директориями, а также была реализована программа, которая рекурсивно обходит файловую систему для поиска конкретных файлов или строк в ней.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

#define MAX_INCLUDES 20000
#define MAX_PATH_LEN 1024
#define INITIAL_PATH "./labyrinth"
#define INITIAL_FILE "file.txt"
#define OUTPUT_FILE "result.txt"

char *deadlocks[MAX_INCLUDES];
int deadlock_count = 0;
char trace[MAX_INCLUDES][MAX_PATH_LEN];
int trace_count = 0;

void add_deadlock(const char *deadlock) {
    for (int i = 0; i < deadlock_count; ++i) {
        if (strcmp(deadlocks[i], deadlock) == 0) {
            return;
        }
    }
    deadlocks[deadlock_count] = (char *)malloc((strlen(deadlock) + 1)
* sizeof(char));
    strcpy(deadlocks[deadlock_count], deadlock);
    deadlock_count++;
}

void get_deadlocks(const char *path) {
    DIR *dir = opendir(path);
    if (!dir) return;

    struct dirent *entry;
    while ((entry = readdir(dir))) {
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name,
"..") == 0)
            continue;

        char fullpath[MAX_PATH_LEN];
        snprintf(fullpath,   sizeof(fullpath),   "%s/%s",   path,
entry->d_name);

        if (entry->d_type == 4) {
            get_deadlocks(fullpath);
            get_deadlocks(fullpath);
        } else {
            FILE *file = fopen(fullpath, "r");
            if (!file) continue;

            char line[MAX_PATH_LEN], file_include[MAX_PATH_LEN];
            int include_count = 0;
```

```
                while (fgets(line, sizeof(line), file)) {
                    if (strcmp(line, "Deadlock") == 0) {
                        add_deadlock(entry->d_name);
                        break;
                    }
                    if (strcmp(line, "Minotaur") == 0) {
                        include_count++;
                        break;
                    }
                    if (sscanf(line, "@include %s", file_include) == 1)
{
                        int isDeadlock = 0;
                        for (size_t i = 0; i < deadlock_count; i++) {
                            if (strcmp(file_include, deadlocks[i]) == 0)
{
                                isDeadlock = 1;
                                break;
                            }
                        }
                        if (!isDeadlock) include_count++;
                    }
                }

                if (include_count == 0) {
                    add_deadlock(entry->d_name);
                }
                fclose(file);
            }
        }
        closedir(dir);
    }

    void process_directory(const char *path, const char *filename) {
        DIR *dir = opendir(path);
        if (!dir) return;

        struct dirent *entry;
        while ((entry = readdir(dir))) {
            if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name,
"..") == 0)
                continue;

            char fullpath[MAX_PATH_LEN];
            snprintf(fullpath,   sizeof(fullpath),   "%s/%s",   path,
entry->d_name);

            if (entry->d_type == 4)
                process_directory(fullpath, filename);
            else if (strcmp(entry->d_name, filename) == 0) {
                FILE *file = fopen(fullpath, "r");
                if (!file) continue;

                char line[MAX_PATH_LEN], file_include[MAX_PATH_LEN];
                while (fgets(line, sizeof(line), file)) {
                    if (strcmp(line, "Minotaur") == 0) {
                        FILE *output_file = fopen(OUTPUT_FILE, "w");
                        for (int i = 0; i < trace_count; i++) {
```

```
                            fprintf(output_file, "%s\n", trace[i]);
                        }
                        fprintf(output_file, "%s", fullpath);
                        fclose(output_file);
                        return;
                    }
                    if (sscanf(line, "@include %s", file_include) == 1)
{
                        int isDeadlock = 0;
                        for (size_t i = 0; i < deadlock_count; i++) {
                            if (strcmp(file_include, deadlocks[i]) == 0)
{
                                isDeadlock = 1;
                                break;
                            }
                        }
                        if (!isDeadlock) {
                            strcpy(trace[trace_count++], fullpath);
                            process_directory(INITIAL_PATH,
file_include);
                        }
                    }
                }
                fclose(file);
            }
        }
        closedir(dir);
    }

    int main() {
        get_deadlocks(INITIAL_PATH);
        get_deadlocks(INITIAL_PATH);
        process_directory(INITIAL_PATH, INITIAL_FILE);
        return 0;
    }
```