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. Вариант 1

Студент гр. 3343 _____ Коршков А.А.

Преподаватель _____ Государкин Я.С.

Санкт-Петербург
2024

Цель работы

Научиться работать с рекурсивными функциями, создавать их, работать с директориями, с файлами, обрабатывать их содержимое, записывать результат работы программы в файл.

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. (@include <test.txt>)

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

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.

Выполнение работы

Сначала выделяем память под запись всех путей, которые приведёт к файлу-минотавру. С помощью функции find_file определяем, относительный путь к нашему 1-ому файлу (file.txt). В нём мы создаём переменную full_path_name, в которой будет храниться этот путь. Открываем самую верхнюю директорию (labyrinth), затем мы создаём структуру de и с помощью функции readdir пробегаемся по нашей папке. Если файл был найден (тип файла DT_REG, strcmp(de->d_name, filename) == 0), то с помощью pathcat происходит соединение записанной директории и имени найденного файла, и в этом случае завершаем цикл while. Если была найдена директория, то соединяется сохранённый путь и имя этой директории, а в full_path_name записывается результат функции find_file, в который подаётся новая директория и имя того же файла, который необходимо найти (происходит рекурсия).

На вход функции check_file подаётся директория, в которой нужно провести поиск и файл file.txt, после чего файл открывается и читается построчно.

Если в файле есть строка @include ..., то программа проверяет, вернёт ли рекурсивная функция 1 или нет. Единица гарантирует, что этот файл необходим для нахождения файла-минотавра.

Если в файле находиться слово «Deadlock» (тупик), то программа завершает цикл, закрывает файл и возвращает 0.

Если в файле находится «Minotaur», то программа будет возвращать 1. После чего каждый предыдущий путь до этого файла будет записываться, пока не закончаться.

В функции main также записывается путь до самого первого файла file.txt.

Т.к. функция будет записывать результат в переменную result начиная с найденного файла, то необходимо вывести пути в обратном порядке. Создаём динамический массив result_table, с помощью strtok разбиваем текст на предложения и записываем в result_table. Затем открываем result.txt в режиме

записи (создаём новый файл) и в обратном порядке записываем в него пути до файла из динамического массива.

Выводы

Была написана программа, которая рекурсивно проходит по файлам, находит «Minotaur» и возвращает путь из файлов до него. Изучен способ открытия директорий и файлов, как создавать и записывать информацию в файлы txt.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

#define SIZE_LINE 250
#define DEFAULT_DIR "labyrinth"
#define FINAL_WORD "Minotaur"
#define DEAD_WORD "Deadlock"

char *pathcat(const char *path1, const char *path2)
{
    int res_path_len = strlen(path1) + strlen(path2) + 2; // определение
длины новой строки с учетом символов / и символа конца строки
    char *res_path = malloc(res_path_len * sizeof(char)); // выделение
памяти под новую строку
    sprintf(res_path, "%s/%s", path1, path2);             // форматный
вывод данных в строку return res_path;
    return res_path;
}

char *find_file(const char *dir_name, const char *filename)
{
    char *full_path_file = NULL; // изначально файл не найден
    DIR *dir = opendir(dir_name);
    if (dir)
    {
        struct dirent *de = readdir(dir);
        while (de)
        {
            if (de->d_type == DT_REG && !strcmp(de->d_name, filename))
            {
                // файл найден
                full_path_file = pathcat(dir_name, filename);
            }
            else if (de->d_type == DT_DIR && strcmp(de->d_name, ".") != 0
&& strcmp(de->d_name, "..") != 0)
            {
                char *new_dir = pathcat(dir_name, de->d_name);
                // запись результата поиска во вложенной директории
                full_path_file = find_file(new_dir, filename);
                free(new_dir);
            }
            if (full_path_file) // файл найден, завершение поиска
                break;
            de = readdir(dir);
        }
        closedir(dir);
    }
    else
        printf("Failed to open %s directory\n", dir_name);
```

```
    return full_path_file;
}

int check_file(const char *file_path, char **result)
{
    FILE *file = fopen(file_path, "r"); // открытие файла на чтение
    if (!file)
    {
        printf("Failed to open %s file\n", file_path);
        exit(0);
    }
    char data[SIZE_LINE];
    char *read_result;
    while ((read_result = fgets(data, SIZE_LINE, file)) != NULL)
    {
        if (strcmp(data, FINAL_WORD) == 0)
        {
            fclose(file);
            return 1;
        }
        else if (strcmp(data, DEAD_WORD) == 0)
        {
            break;
        }
        else
        {
            sscanf(data, "@include %s", read_result);
            if (check_file(find_file(DEFAULT_DIR, data), result))
            {
                strcat(*result, "./");
                strcat(*result, find_file(DEFAULT_DIR, data));
                strcat(*result, "\n");
                fclose(file);
                return 1;
            }
        }
    }
    fclose(file);
    return 0;
}

int main()
{
    char *result = malloc(sizeof(char) * 1000);
    char *file = find_file(DEFAULT_DIR, "file.txt");
    check_file(file, &result);
    strcat(result, "./");
    strcat(result, file);
    strcat(result, "\n");
    int n = 0;
    char **result_table = malloc(sizeof(char *));
    char *res = strtok(result, "\n");
    while (res != NULL)
    {
        result_table[n++] = res;
        result_table = realloc(result_table, sizeof(char *) * (n + 1));
        res = strtok(NULL, "\n");
```

```
    }
    FILE *res_file = fopen("result.txt", "w");

    for (int i = n - 1; i > -1; i--)
    {
        if (i == 0)
        {
            fprintf(res_file, "%s", result_table[i]);
        }
        else
        {
            fprintf(res_file, "%s\n", result_table[i]);
        }
    }
    fclose(res_file);
    free(result);
    free(result_table);
    return 0;
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Тест № 1.

Входные данные:

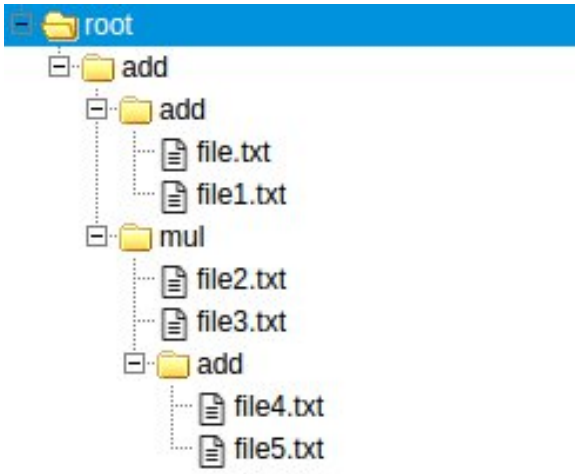

Рисунок 1 – Структура директории labyrinth (Тест №1)

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

Выходные данные (result.txt):

./labyrinth/add/add/file.txt

./labyrinth/add/mul/add/file4.txt

./labyrinth/add/mul/file2.txt

./labyrinth/add/mul/file3.txt

Тест № 2.

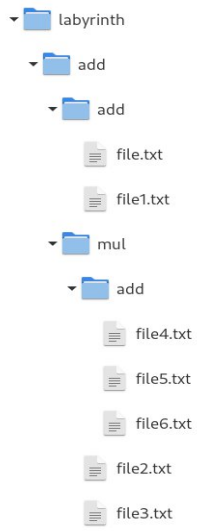

Рисунок 2 – Структура директории labyrinth (Тест №2)

file.txt:

@include file1.txt

@include file4.txt

@include file5.txt

file1.txt:

@include file4.txt

file2.txt:

@include file6.txt

file3.txt:

Minotaur

file4.txt:

Deadlock

file5.txt:

@include file2.txt

@include file3.txt

file6.txt:

Deadlock

Выходные данные (result.txt):

./labyrinth/add/add/file.txt

./labyrinth/add/mul/add/file5.txt

./labyrinth/add/mul/file3.txt

Примечание:

Файлы file4 и file6 – тупики, file3.txt – файл-минотавр. В файле file.txt есть ссылки на txt файлы file1, file4 и file5. File1.txt содержит в себе ссылку на file4, который тупик (т.е. первые 2 ссылки в file.txt являются тупиками). File5.txt содержит ссылки на file2.txt и file3.txt. File2.txt ссылается на file6.txt, который в свою очередь является тупиком. File3.txt – конечный файл, поэтому функция check_file будет возвращать 1, чтобы правильно вывести пути нужных файлов.

Этот пример показывает, что программа будет выводить только один правильный путь, насколько бы глубоко не находился файл-тупик или если бы на него ссылались несколько файлов.