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. 3343 _____ Пименов П.В.

Преподаватель _____ Государкин Я.С.

Санкт-Петербург
2024

Цель работы

Изучить общий принцип работы рекурсивных алгоритмов, написать программу на языке C, выполняющую обход файловой системы.

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. 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

Описание функций

1. *char* read_file(char* path)* – считывает текст из файла, возвращает указатель на считанную строку
2. *char* strsepcat(char* left, char* sep, char* right)* – соединяет три строки в одну (в порядке расположения аргументов), возвращает указатель на новую строку
3. *char* create_path(char* path, char* step)* – создает новый путь к файлу (строку), возвращает на нее указатель

4. *char* create_track(char* old_track, char* step)* – создает новую цепочку файлов с путями (строку), возвращает на нее указатель
5. *char* find_file(char* root_path, char* search_path, char* target_file)* – осуществляет рекурсивный поиск файла по имени

Изначально программа ищет файл с именем «file.txt». Считывается его текст, если это файл-минотавр, то возвращается путь до него, если же он содежит «@include …», то запускается поиск файлов по имени, содержащихся в этих «включениях». В итоге возвращается функция возвращает полную цепочку файлов с путями до файла-минотавра, которая записывает эту цепочку в файл «result.txt».

Разработанный программный код см. в приложении А.

Выводы

Был изучен общий принцип работы рекурсивных алгоритмов, написана программа на языке C, выполняющая обход файловой системы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

#define MEMORY_BLOCK_SIZE 10
#define ROOT_DIR          "./labyrinth"
#define ROOT_FILE         "file.txt"
#define OUTPUT_FILE       "result.txt"

char* read_file(char* path);
char* strsepcat(char* left, char* sep, char* right);
char* create_path(char* path, char* step);
char* create_track(char* old_track, char* step);
char* find_file(char* root_path, char* search_path, char*
target_file);

char*
read_file(char* path) {
    FILE* file = fopen(path, "r");
    if (!file) {
        return NULL;
    }
    size_t size = 0, capacity = MEMORY_BLOCK_SIZE;
    char* content = (char*)malloc(sizeof(char) * capacity);
    int last_char = getc(file);
    if (last_char == EOF || last_char == '\0') {
        fclose(file);
        free(content);
        content = NULL;
        return NULL;
    }
    content[size++] = last_char;
    content[size] = '\0';
    last_char = getc(file);
    while (last_char != EOF && last_char != '\0') {
        content[size++] = last_char;
        content[size] = '\0';
        if (size == capacity - 1) {
            capacity += MEMORY_BLOCK_SIZE;
            content = (char*)realloc(content, capacity);
        }
        last_char = getc(file);
    }
    fclose(file);
    return content;
}
```

```
char*
strsepcat(char* left, char* sep, char* right) {
    char* new = (char*)malloc(sizeof(char) * (strlen(left) +
strlen(sep) + strlen(right) + 1));
    sprintf(new, "%s%s%s", left, sep, right);
    return new;
}

char*
create_path(char* path, char* step) {
    return strsepcat(path, "/", step);
}

char*
create_track(char* old_track, char* step) {
    return strsepcat(step, "\n", old_track);
}

char*
find_file(char* root_path, char* search_path, char* target_file) {
    char* result = NULL;
    DIR* current_dir = opendir(search_path);
    if (!current_dir) {
        return result;
    }
    struct dirent* entry;
    while ((entry = readdir(current_dir))) {
        if (entry->d_type == DT_REG) {
            if (!strcmp(entry->d_name, target_file)) {
                char* file_path = create_path(search_path,
target_file);
                char* text = read_file(file_path);
                if (!strcmp(text, "Minotaur")) {
                    result = strdup(file_path);
                    free(file_path);
                    file_path = NULL;
                    free(text);
                    text = NULL;
                    break;
                } else if (strstr(text, "@include ")) {
                    char* text_duplicate = strdup(text);
                    char* msc;
                    char* line = strtok_r(text_duplicate, "\n", &msc);
                    char* node_file_name = NULL;
                    char* node_result = NULL;
                    while (line && !node_result) {
                        node_file_name = strstr(line, " ") + 1 *
sizeof(char);
                        node_result = find_file(root_path, root_path,
node_file_name);
                        line = strtok_r(NULL, "\n", &msc);
                    }
                    if (node_result) {
                        result = create_track(node_result, file_path);
                        free(text_duplicate);
                        text_duplicate = NULL;
```

```
                        free(node_result);
                        node_result = NULL;
                        free(file_path);
                        file_path = NULL;
                        free(text);
                        text = NULL;
                        break;
                    }
                    free(text_duplicate);
                    text_duplicate = NULL;
                }
                //случай с Deadlock или мусором автоматически
отбрасывается
                free(file_path);
                file_path = NULL;
                free(text);
                text = NULL;
                break;
            }
        } else if (entry->d_type == DT_DIR) {
            if (!strcmp(entry->d_name, ".") || !strcmp(entry->d_name,
"..")) {
                continue;
            }
            char* new_path = create_path(search_path, entry->d_name);
            char* next_dir = find_file(root_path, new_path,
target_file);
            free(new_path);
            new_path = NULL;
            if (next_dir) {
                result = next_dir;
                break;
            }
        }
    }
    closedir(current_dir);
    return result;
}

int
main() {
    char* result_track = find_file(ROOT_DIR, ROOT_DIR, ROOT_FILE);
    FILE* output_file = fopen(OUTPUT_FILE, "w");
    if (output_file) {
        fprintf(output_file, "%s", result_track);
    }
    fclose(output_file);
    free(result_track);
    result_track = NULL;
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

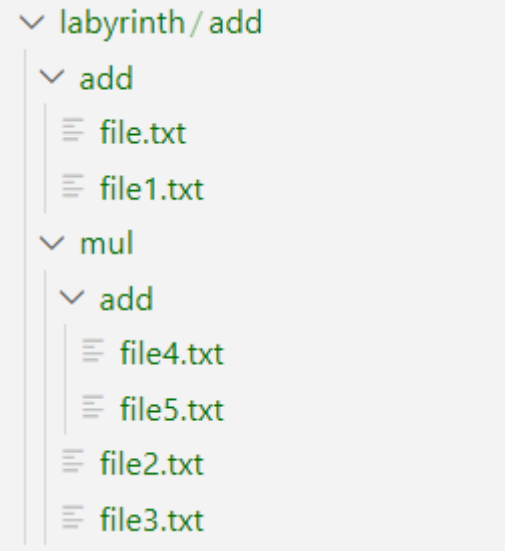

Рисунок 1 – Расположение файлов для теста

Содержимое файлов:

- 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

Результат работы программы: содержимое файла «result.txt»

./labyrinth/add/add/file.txt

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

./labyrinth/add/mul/file2.txt

./labyrinth/add/mul/file3.txt

Примечание: программа работает корректно.