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.

Студентка гр. 3343 _____ Синицкая Д.В.

Преподаватель _____ Государкин Я. С.

Санкт-Петербург

2024

Цель работы

Освоение работы с рекурсивными функциями и файловой системой, в языке программирования Си. Создание программы, выполняющей рекурсивный обход файловой системы для поиска файла, содержащего заданное слово.

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. 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

Функция char *pathcat(const char *path1, const char *path2) нужна для создания путей. Она принимает две строки (пути к файлам), объединяет их и возвращает новую строку-путь.

Функция char *find_file(const char *dir_name, const char *filename) нужна для поиска файла в директории. Она рекурсивно ищет файл с заданным именем в указанной директории. Если файл найден, возвращается его полный путь.

Функция int working_with_a_file(const char *file_path, char **result) нужна для работы с файлом. Она открывает файл, читает содержимое строки за строкой. Если встречает определенные ключевые слова ("Minotaur", "@include"), то выполняет определенные действия.

В функции main() происходит основная логика программы. Она ищет файл "file.txt" в директории "labyrinth", обрабатывает его содержимое с помощью `working_with_a_file`, и записывает результат в файл "result.txt".

Таким образом, код реализует задачу обработки файлов и директорий, включая рекурсивный поиск и обработку содержимого файлов внутри других файлов (через директиву `@include`).

Разработанный программный код см. в приложении А.

Выводы

В процессе выполнения лабораторной работы мной были освоены навыки работы с рекурсивными функциями, директориями и файловой системой. Были изучены необходимые языковые конструкции и особенности написания рекурсивных функций, функции для работы с файлами и директориями языка Си.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

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

//функция для создания путей
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;
}

//функция поиска файла в директории
char *find_file(const char *dir_name, const char *filename)
{
    char *full_path_file = NULL; //изначально файл не найден
    DIR *dir = opendir(dir_name); //открытие директории
    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); //закрытие директории
    }
    else
        printf("Failed to open %s directory\n", dir_name);
    return full_path_file;
}

//функция работы с файлом
int working_with_a_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 information[500];
    char *read_result = fgets(information, 500, file);
    while (read_result != NULL)
    {
          if (strcmp(information, "Minotaur") == 0) //поиск слова
Minotaur
        {
            fclose(file); //закрытие файла
            return 1; //выход из функции
        }
```

```
            else if (strcmp(information, "Deadlock") == 0) //поиск
тупика
        {
            break; //выход из цикла
        }
        else
        {
            //считывание строки, которая начинается с @include
                  sscanf(information, "@include %s", read_result);
//считывание названия другого файла из ссылки
                 char *file_from_the_link = find_file("labyrinth",
information); //поиск файла
            if (working_with_a_file(file_from_the_link, result))
            {
                    strcat(*result, "./"); //запись корня текущего
каталога
                     strcat(*result, file_from_the_link); //запись
полного пути к файлу
                   strcat(*result, "\n"); //запись символа переноса
строки
                free(file_from_the_link); //очистка памяти
                fclose(file); //закрытие файла
                return 1; //выход из функции
            }
            free(file_from_the_link); //очистка памяти
        }

        read_result = fgets(information, 500, file);
    }
    fclose(file);
    return 0;
}


int main()
{
     char *result = malloc(sizeof(char) * 3000); //выделение памяти
под результат
```

```
     char *file = find_file("labyrinth", "file.txt"); //поиск файла в директории
        working_with_a_file(file, &result); //проверка содержимого файла
    strcat(result, "./"); //запись корня текущего каталога
    strcat(result, file); //запись файла
    strcat(result, "\n"); //запись символа переноса строки

    int counter = 0; //счётчик
    //объединение данных для последующей записи в файл
    char **result_for_writing_to_a_file = malloc(sizeof(char *));
    char *token = strtok(result, "\n");
    while (token != NULL)
    {
        result_for_writing_to_a_file[counter++] = token;
                                    result_for_writing_to_a_file = realloc(result_for_writing_to_a_file, sizeof(char *) * (counter + 1));
        token = strtok(NULL, "\n");
    }

    //запись результата в файл
    FILE *res_file = fopen("result.txt", "w");
    for (int i = counter - 1; i > -1; i--)
    {
        if (i == 0)
        {
                                            fprintf(res_file, "%s", result_for_writing_to_a_file[i]);
        }
        else
        {
                                        fprintf(res_file, "%s\n", result_for_writing_to_a_file[i]);
        }
    }

    fclose(res_file); //закрытие файла с результатом
    free(result); //очистка памяти
```

```
    free(result_for_writing_to_a_file); //очистка памяти
    free(file); //очистка памяти
    return 0;
}
```