**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. 3344 |  | Охрименко Д.И. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Изучить возможности работы с файловой системой с использованием языка программирования C, освоить рекурсивные алгоритмы обхода дерева файлов. Применить на практике алгоритм поиска данных в файлах и сохранить результат в новом файле.

## 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.

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.

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

## Выполнение работы

Подключаем заголовочные файлы: <stdio.h>, <stdlib.h>, <string.h> и <dirent.h>.

Вводим макроопределения: SIZE для определения размера считываемых символов из файла, MAXFILES – кол-во файлов не превышает это число.

Функция inDirect() выполняет рекурсивный поиск файла в директориях для определения полного пути к указанному файлу. Параметры: Название директории и имя файла. Возвращаемое значение: Полный путь к найденному файлу.

Функция searchWord() осуществляет поиск ключевых слов "Minotaur" и "Deadlock" в файле, а также рекурсивно обрабатывает вложенные файлы. Параметры: Имя файла и массив строк для записи результатов. Назначение: Считывание информации из файла, поиск ключевых слов, запись путей к файлам, содержащим ключевое слово "Minotaur".

Функция main() вызов функции searchWord() для начала поиска ключевых слов, а затем запись результатов в файл "result.txt".

**Тестирование**

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 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  file5.txt: Deadlock  ./labyrinth/file.txt  ./labyrinth/file1.txt  ./labyrinth/mul/file2.txt  ./labyrinth/mul/file3.txt  ./labyrinth/mul/mul/file4.txt | ./labyrinth/file.txt  ./labyrinth/mul/mul/file4.txt  ./labyrinth/mul/file2.txt  ./labyrinth/mul/file3.txt | Данные обработаны корректно. |

## Выводы

В ходе лабораторной работы изучил основы работы с файловой системой и применил для обхода рекурсивный алгоритм. Программа успешно обошла директорию и сохранила результат в соответствующий файл.

# Приложение А Исходный код программы

Название файла: lb3.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#define SIZE 256

#define MAXFILES 3000

int flag = 0;

int count = 0;

char \*listDir(char\* startDir, const char \*fileName)

{

DIR \*dir;

struct dirent \*de;

char \*full\_path\_file = NULL;

dir = opendir(startDir);

if(dir)

{

de = readdir(dir);

while(de)

{

if (de->d\_type == DT\_REG && !strcmp(de->d\_name, fileName))

{

int Len = strlen(startDir) + strlen(fileName) + 2;

full\_path\_file = malloc(Len \* sizeof(char));

sprintf(full\_path\_file, "%s/%s", startDir, fileName);

break;

}

else if (de->d\_type == DT\_DIR && strcmp(de->d\_name, ".") && strcmp(de->d\_name, ".."))

{

char \*newDirect = malloc(sizeof(char) \* (strlen(startDir) + strlen(de->d\_name) + 2));

sprintf(newDirect, "%s/%s", startDir, de->d\_name);

full\_path\_file = listDir(newDirect, fileName);

free(newDirect);

if (full\_path\_file)

{

break;

}

}

de = readdir(dir);

}

closedir(dir);

}

return full\_path\_file;

}

void searchWord(char\* filename, char\*\* result)

{

char \*file\_path = listDir(".", filename);

FILE \*file = fopen(file\_path, "r");

if (!file)

return;

char data[SIZE];

while(fgets(data, SIZE, file) != NULL && flag == 0)

{

if(strstr(data, "Deadlock"))

{

return;

}

else if(strstr(data, "Minotaur"))

{

flag = 1;

result[count] = malloc(sizeof(char) \* SIZE);

strcpy(result[count++], file\_path);

}

else if(strncmp(data, "@include ", 9) && data[strlen(data) - 1] == '\n')

{

data[strlen(data) - 1] = '\0';

memmove(&data[0], &data[9], sizeof(char) \* SIZE);

searchWord(data, result);

if(flag)

{

result[count] = malloc(SIZE \* sizeof(char));

strcpy(result[count++], file\_path);

}

}

}

fclose(file);

}

int main()

{

char\*\* result = (char\*\*)malloc(sizeof(char\*) \* MAXFILES);

searchWord("file.txt", result);

FILE \*result\_file = fopen("result.txt", "w");

if(result\_file == NULL)

return 1;

for (int i = count - 1; i >= 0; --i)

{

fprintf(result\_file, "%s\n", result[i]);

}

fclose(result\_file);

for (int i = 0; i < count; ++i)

{

free(result[i]);

}

free(result);

return 0;

}