**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. 3342 |  | Колесниченко М.А. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Целью данной лабораторной работы является создание программы на языке C для поиска файла-минотавра в структуре файловой системы. Программа должна рекурсивно обходить директории, начиная с корневой директории "labyrinth", и находить файл с именем "file.txt". Затем она должна анализировать содержимое этого файла и, если обнаруживает строку "Minotaur", записывать путь к этому файлу вместе с цепочкой всех файлов, которые привели к обнаружению файла-минотавра.

## 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.

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

1. Программа начинает свою работу с поиска файла "file.txt" в корневой директории "labyrinth".
2. После нахождения файла "file.txt" программа анализирует его содержимое.
3. Если программа обнаруживает строку "Minotaur", она записывает путь к этому файлу вместе с цепочкой всех файлов, которые привели к обнаружению файла-минотавра.
4. Для обработки ссылок на другие файлы программа рекурсивно ищет их в структуре файловой системы, начиная с корневой директории.
5. Программа продолжает свой поиск до тех пор, пока не будет обнаружен файл-минотавр или не будет достигнут конец структуры файловой системы.
6. Результат работы программы записывается в файл "result.txt", который находится в директории /home/box, как требуется по условию задачи.

Разработанный программный код см. в приложении А.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |
| --- | --- | --- |
| № п/п | Входные данные | Выходные данные |
|  | 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 | ./root/add/add/file.txt  ./root/add/mul/add/file4.txt  ./root/add/mul/file2.txt  ./root/add/mul/file3.txt |

## Выводы

Была разработана программа на языке C, которая эффективно обходит структуру файловой системы, начиная с заданной корневой директории "labyrinth".

# Приложение А Исходный код программы

Название файла: main.c

#include <linux/limits.h>

#include <dirent.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>

#include <regex.h>

#include <stdio.h>

#define BUFFER 1024

#define LABYRINTH\_DIR "labyrinth"

#define RESULT\_FILE "result.txt"

#define INCLUDE\_REGEX "^\\@include (\\w+\\.\\w+)\n$"

void freeAnswer(char\*\* answer, int pathCounter) {

for (int i = 0; i < pathCounter; ++i) {

free(answer[i]);

}

free(answer);

}

int isInclude(const char \*filename, regmatch\_t matchptr[], size\_t nmatch) {

regex\_t regexComp;

if (regcomp(&regexComp, INCLUDE\_REGEX, REG\_EXTENDED)) {

return 0;

}

int result = regexec(&regexComp, filename, nmatch, matchptr, 0);

regfree(&regexComp);

return result == 0;

}

void findFile(const char \*startDir, const char \*filename, char \*pathToFile) {

DIR \*dir = opendir(startDir);

if (!dir) {

return;

}

struct dirent \*de;

while ((de = readdir(dir)) != NULL) {

if (de->d\_type == DT\_REG && strcmp(de->d\_name, filename) == 0) {

snprintf(pathToFile, PATH\_MAX, "%s/%s", startDir, de->d\_name);

closedir(dir);

return;

}

if (de->d\_type == DT\_DIR && strcmp(de->d\_name, ".") != 0 && strcmp(de->d\_name, "..") != 0) {

char nextDir[PATH\_MAX];

snprintf(nextDir, PATH\_MAX, "%s/%s", startDir, de->d\_name);

findFile(nextDir, filename, pathToFile);

}

}

closedir(dir);

}

int isDeadlock(const char \*filename) {

FILE \*f = fopen(filename, "r");

if (!f) {

return 1;

}

char line[BUFFER];

fgets(line, BUFFER, f);

if (strstr(line, "Deadlock")) {

fclose(f);

return 1;

}

fclose(f);

return 0;

}

int check(const char \*pathToFile, char \*\*answer, int \*pathCounter) {

FILE \*f = fopen(pathToFile, "r");

if (!f) {

return 0;

}

char line[BUFFER];

regmatch\_t matchptr[2];

int x = 0;

while (fgets(line, BUFFER, f)) {

if (strstr(line, "Minotaur")) {

answer[\*pathCounter] = malloc(strlen(pathToFile) + 1);

if (!answer[\*pathCounter]) {

perror("Memory allocation error");

return 0;

}

strcpy(answer[\*pathCounter], pathToFile);

(\*pathCounter)++;

fclose(f);

return 1;

} else if (isInclude(line, matchptr, 2) && x != 1) {

char newFileName[BUFFER] = {0};

char pathToNewFile[PATH\_MAX] = {0};

strcat(newFileName, &line[matchptr[1].rm\_so]);

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

findFile(LABYRINTH\_DIR, newFileName, pathToNewFile);

if (x != 1 && !isDeadlock(pathToNewFile)) {

x = check(pathToNewFile, answer, pathCounter);

}

}

}

if (x == 1) {

answer[\*pathCounter] = malloc(strlen(pathToFile) + 1);

strcpy(answer[\*pathCounter], pathToFile);

(\*pathCounter)++;

}

fclose(f);

return x;

}

int main() {

char start[PATH\_MAX] = "";

FILE \*fp = fopen(RESULT\_FILE, "w");

if (!fp) {

perror("Error opening file");

return 1;

}

int pathCounter = 0;

char \*\*answer = malloc(sizeof(char \*) \* 1000);

if (!answer) {

perror("Memory allocation error");

fclose(fp);

return 1;

}

findFile(LABYRINTH\_DIR, "file.txt", start);

if (!check(start, answer, &pathCounter)) {

printf("Minotaur not found in labyrinth.\n");

fclose(fp);

freeAnswer(answer, pathCounter);

return 1;

}

for (int i = pathCounter - 1; i >= 0; i--) {

fprintf(fp, "./%s\n", answer[i]);

}

fclose(fp);

freeAnswer(answer, pathCounter);

return 0;

}