**Практическая работа №7. Рекурсивный поиск слова в файлах директории**

**Описание**

Программа wordsearch на языке C рекурсивно обходит заданную директорию и её вложенные каталоги (включая скрытые) и ищет указанное слово во всех текстовых файлах, используя mmap для чтения содержимого. При обнаружении выводятся полный путь к файлу, номер строки и сама строка с найденным словом.

**Требования**

* Директория поиска передаётся первым аргументом командной строки (по умолчанию ~/files).
* Слово для поиска передаётся вторым аргументом командной строки.
* Поддерживается флаг -i для игнорирования регистра.
* Обрабатываются скрытые файлы и директории.
* Корректная обработка ошибок (отсутствие директории, ошибки доступа).

**Makefile**

CC = gcc

CFLAGS = -std=c99 -Wall -Wextra

TARGET = wordsearch

SRC = wordsearch.c

.PHONY: all clean

all: $(TARGET)

$(TARGET): $(SRC)

$(CC) $(CFLAGS) -o $(TARGET) $(SRC)

clean:

rm -f $(TARGET)

**Листинг исходного кода с комментариями**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <dirent.h>

#include <sys/stat.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <unistd.h>

#include <errno.h>

#include <limits.h>

#ifndef PATH\_MAX

#define PATH\_MAX 4096 // дефолтная максимальная длина пути, если не задана системой

#endif

#define LINE\_BUF\_SIZE 8192 // максимальный размер строки для поиска

static const char \*search\_word; // искомое слово

static int ignore\_case = 0; // флаг игнорирования регистра

// Функция поиска подстроки в строке с учётом/без учёта регистра

char \*find\_substr(const char \*haystack, const char \*needle) {

if (!ignore\_case)

return strstr(haystack, needle); // простое вхождение

size\_t nl = strlen(needle);

for (const char \*p = haystack; \*p; ++p) {

size\_t i;

for (i = 0; i < nl; ++i) {

if (tolower((unsigned char)p[i]) != tolower((unsigned char)needle[i]))

break;

}

if (i == nl)

return (char\*)p;

}

return NULL;

}

// Поиск во вложенном файле через mmap

void search\_file(const char \*filepath) {

int fd = open(filepath, O\_RDONLY); // открываем файл

if (fd < 0) return; // на ошибки доступа — пропускаем

struct stat st;

if (fstat(fd, &st) < 0 || st.st\_size == 0) { // проверка размера

close(fd);

return;

}

// отображаем файл в память

char \*data = mmap(NULL, st.st\_size, PROT\_READ, MAP\_PRIVATE, fd, 0);

if (data == MAP\_FAILED) {

close(fd);

return;

}

unsigned long lineno = 1;

size\_t line\_start = 0;

// перебираем байты и ищем границы строк

for (size\_t i = 0; i <= (size\_t)st.st\_size; ++i) {

if (i == (size\_t)st.st\_size || data[i] == '\n') {

size\_t len = i - line\_start;

char \*line = malloc(len + 1);

memcpy(line, data + line\_start, len);

line[len] = '\0';

// если нашли вхождение — выводим

if (find\_substr(line, search\_word)) {

printf("%s:%lu: %s\n", filepath, lineno, line);

}

free(line);

line\_start = i + 1;

lineno++;

}

}

munmap(data, st.st\_size);

close(fd);

}

// Рекурсивный обход директории

void process\_dir(const char \*path) {

DIR \*d = opendir(path);

if (!d) return;

struct dirent \*entry;

while ((entry = readdir(d)) != NULL) {

if (strcmp(entry->d\_name, ".") == 0 || strcmp(entry->d\_name, "..") == 0)

continue;

char fullpath[PATH\_MAX];

snprintf(fullpath, sizeof(fullpath), "%s/%s", path, entry->d\_name);

struct stat st;

if (stat(fullpath, &st) == -1) continue;

if (S\_ISDIR(st.st\_mode))

process\_dir(fullpath);

else if (S\_ISREG(st.st\_mode))

search\_file(fullpath);

}

closedir(d);

}

int main(int argc, char \*argv[]) {

const char \*dir = NULL;

int argi = 1;

// обработка ключа -i

if (argc > 1 && strcmp(argv[1], "-i") == 0) {

ignore\_case = 1;

argi++;

}

// читаем директорию из аргументов или по умолчанию ~/files

if (argi < argc)

dir = argv[argi++];

else {

const char \*home = getenv("HOME");

if (!home) {

fprintf(stderr, "Не задана директория и переменная HOME не установлена.\n");

return EXIT\_FAILURE;

}

static char default\_dir[PATH\_MAX];

snprintf(default\_dir, sizeof(default\_dir), "%s/files", home);

dir = default\_dir;

}

// читаем слово для поиска

if (argi < argc)

search\_word = argv[argi];

else {

fprintf(stderr, "Использование: %s [-i] [директория] слово\n", argv[0]);

return EXIT\_FAILURE;

}

process\_dir(dir);

return EXIT\_SUCCESS;

}

**Пример текстового вывода результатов работы**

$ mkdir -p ~/files/subdir

$ echo -e "Hello world\nExample line" > ~/files/test.txt

$ echo -e "hello World" > ~/files/subdir/other.txt

# Запуск без игнорирования регистра

$ ./wordsearch hello

/home/user/files/test.txt:1: Hello world

# Запуск с -i (игнорирование регистра)

$ ./wordsearch -i ~/files hello

/home/user/files/test.txt:1: Hello world

/home/user/files/subdir/other.txt:1: hello World

**Анализ работы программы**

* **Проблема**: чтение больших файлов через fgets может быть медленным и неудобно обрабатывать бинарные символы.  
  **Решение**: использование mmap для отображения файла в память и прямого побайтового разбора строк.
* **Проблема**: игнорирование регистра при поиске.  
  **Решение**: реализована функция find\_substr, сравнивающая символы через tolower.
* **Проблема**: обход скрытых файлов и директорий (имена с точкой) по умолчанию пропускаются.  
  **Решение**: рекурсивная функция process\_dir не отфильтровывает скрытые элементы, кроме . и .., поэтому все файлы и папки обрабатываются.
* **Проблема**: потенциально длинные строки (более LINE\_BUF\_SIZE).  
  **Решение**: в версии на основе mmap динамически распознаются границы строк без фиксированного буфера, что устраняет ограничение размера строки.