Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет безопасности информационных технологий

Дисциплина:

«Основы системного программирования»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1.1

«Работа с файлами и каталогами»

Вариант 4

Выполнили:

Суханкулиев Мухаммет,

студент группы N3246

(подпись)

Проверил:

Грозов В. А,

преподаватель практики

(отметка о выполнении)

(подпись)

Санкт-Петербург

2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – разработать на языке C для ОС Linux программу, позволяющую выполнять рекурсивный поиск файлов, содержащих заданную последовательность байтов, начиная с указанного каталога. Программа должна использовать `nftw()` для обхода файловой системы.

Постановка задачи:

Необходимо создать консольную утилиту, принимающую аргументы командной строки:

- `каталог` — путь, с которого начинается рекурсивный поиск;
- `цель_поиска` — последовательность байтов в формате `0xhh[hh*]`, где `hh` — две шестнадцатеричные цифры.

Программа должна рекурсивно обходить файловую систему с помощью `nftw()`, проверять содержимое файлов на наличие указанной последовательности байтов и выводить пути найденных файлов. Так же должно быть реализовано:

- Обработка опций `-h` (`--help`), `-v` (`--version`);
- Реализация отладочного режима через переменную окружения `LAB11DEBUG`;
- Программа должна собираться с помощью `make` и соответствовать стандарту C11;
- Компиляция с `-Wall -Wextra -Werror`, без предупреждений;
- Тестирование через `valgrind` на предмет утечек памяти.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Подготовить тестовый набор файлов;
2. Разработать алгоритм поиска файлов по заданной последовательности байтов;
3. Реализовать рекурсивный обход файловой системы с использованием `nftw()`;
4. Реализовать обработку аргументов командной строки (`getopt_long()`);
5. Обеспечить корректное поведение при ошибках (отсутствие доступа, неверные аргументы и т. д.);
6. Реализовать поддержку отладочного режима (`LAB11DEBUG`);
7. Разработать и протестировать `Makefile` (`all`, `clean`);
8. Провести тестирование, включая проверку утечек памяти (`valgrind`);
9. Оптимизировать программу (`-O3`), провести повторное тестирование.

1 РАЗРАБОТКА ПРОГРАММЫ

Программа разработана на языке C с использованием стандарта C17.

1.1 Структура программы

Программа состоит из заголовочного файла `lab11msN3246.h` и исходного файла `lab11msN3246.c`. Основные функции:

- `parse_hex_pattern()` — разбор строки с байтовым шаблоном.
- `search_file()` — обработка файлов, поиск заданной последовательности.
- `print_help()`, `print_version()` — вывод справки и информации о студенте.
- `log_error()` — запись ошибок в `syslog`.

1.2 Обработка аргументов командной строки

Аргументы обрабатываются через `getopt_long()`. Поддерживаются:

- `-h`, `--help` — вывод справки.
- `-v`, `--version` — вывод информации о студенте.
- `<directory>` — начальный каталог.
- `<search_pattern>` — последовательность байтов в формате `0xhh[hh*]`.

1.3 Рекурсивный обход файловой системы

Файлы и каталоги обходятся функцией `nftw()`, обработка выполняется в `search_file()`. Обрабатываются файлы (`FTW_F`), каталоги (`FTW_D`) и другие типы.

1.4 Чтение файлов и поиск последовательности байтов

Файлы читаются блоками (`BUFFER_SIZE`), поиск выполняется с помощью `memcmp()`. При обнаружении совпадения выводится путь к файлу.

1.5 Реализация отладочного режима

При наличии переменной окружения `LAB11DEBUG` выводится дополнительная информация о ходе работы (`fprintf(stderr, "[DEBUG] ...")`).

1.6 Обработка ошибок

Реализована обработка ошибок:

- Неправильный формат аргументов (`fprintf(stderr, "Error: ...")`).
- Ошибки открытия файлов (`perror() + проверка errno`).
- Ошибки работы `nftw()` логируются в `syslog (log_error()`).

1.7 Make-файл

`Makefile` содержит цели:

- `all` — сборка программы.
- `clean` — удаление бинарных файлов.
- Компиляция выполняется с `-Wall -Wextra -Werror -g -O3`.

1.7.1 `Makefile`

```
CC = gcc
CFLAGS = -Wall -Wextra -Werror -g -O3
LDFLAGS =

SRCS = lab11msN3246.c

OBJS = $(SRCS:.c=.o)

APP = lab11msN3246

.PHONY: all clean

all: $(APP)

$(APP): $(OBJS)
	$(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)

%.o: %.c
	$(CC) $(CFLAGS) -c $<

clean:
	@rm -f $(OBJS) $(APP)
```

2 ТЕСТИРОВАНИЕ

2.1 Подготовка тестового набора

Создадим файлы с разными данными, включая тестовые файлы с разной последовательностью байтов.

2.1.1 generate_test_files.sh

```
#!/bin/bash

# Файлы с заданными байтовыми последовательностями
echo -n -e "\x41\x42\x43" > file1.txt  # 'A', 'B', 'C'
echo -n -e "\xc0\xff\xee" > file2.txt  # 192, 255, 238 (in HTML number)
echo -n -e "\x00\x01\x02" > file3.txt  # 0x00, 0x01, 0x02

# Большой файл (10 МБ) с добавлением шаблона
dd if=/dev/zero of=largefile.txt bs=1M count=10 status=none
echo -n -e "\x41\x42\x43" >> largefile.txt

# Файлы с другим байтовым шаблоном
echo -n -e "\x41\x42\x45\xff" > file_with_pattern.txt
echo -n -e "\x41\x42\x45\xff" > another_file_with_pattern.txt
echo -n -e "\x41\x42\x45\xff" > binary_file_with_pattern.txt
echo -n -e "\x41\x42\x45\xff" > binary_file.txt
echo -n -e '\x41\x42\x45\xff' > file.bin

# Бинарные файлы с другим заполнением
dd if=/dev/zero bs=4M count=5 status=none | tr '\000' '\x41' > file1.bin
dd if=/dev/zero bs=1M count=20 status=none | tr '\000' '\x41' > file2.bin

# Создание большого файла с повторяющимся шаблоном
for i in {1..1310720}; do printf '\x41\x42\x45\xff'; done > file3.bin

# Создание тестовой директории с 512 пустыми файлами
mkdir -p ~/Desktop/lab11msN3246/test_dir
for i in {1..512}; do touch ~/Desktop/lab11msN3246/test_dir/file_$i.txt; done

# Создание файла без прав на чтение
touch ~/Desktop/lab11msN3246/file_with_no_read_permission
chmod 000 ~/Desktop/lab11msN3246/file_with_no_read_permission

echo "Тестовые файлы успешно созданы."
```

2.2 Функциональное тестирование

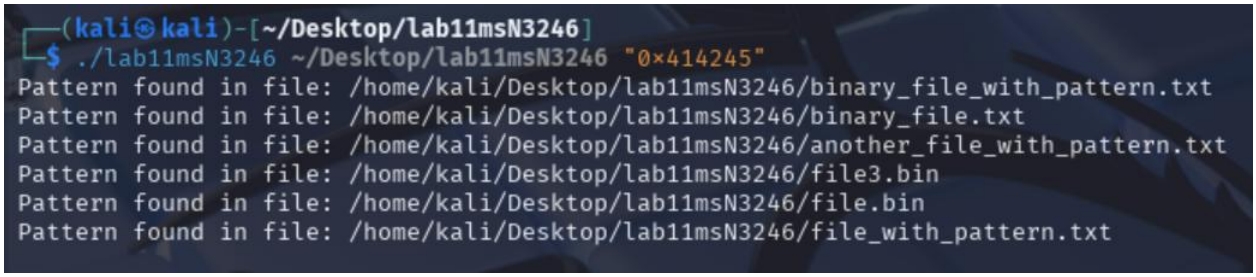

Рисунок 1 – Запуск программы с корректными аргументами

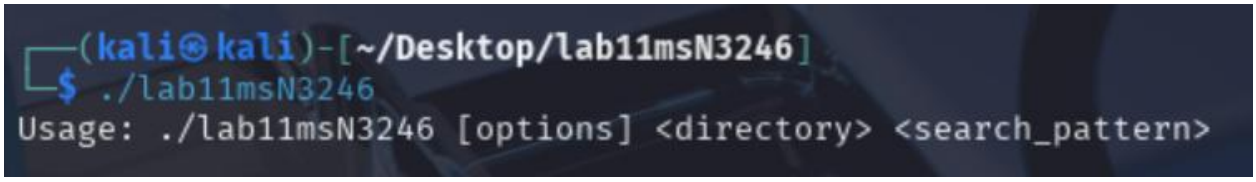

Рисунок 2 – Запуск без аргументов

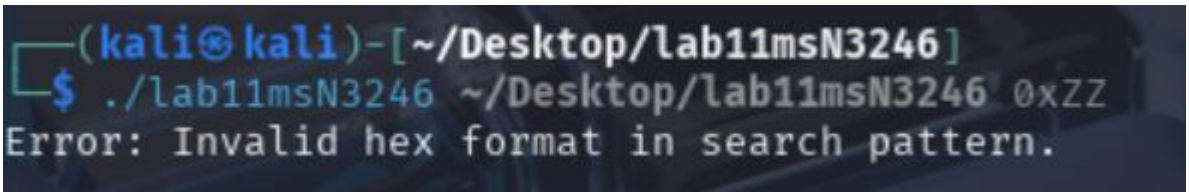

Рисунок 3 – Неверный формат

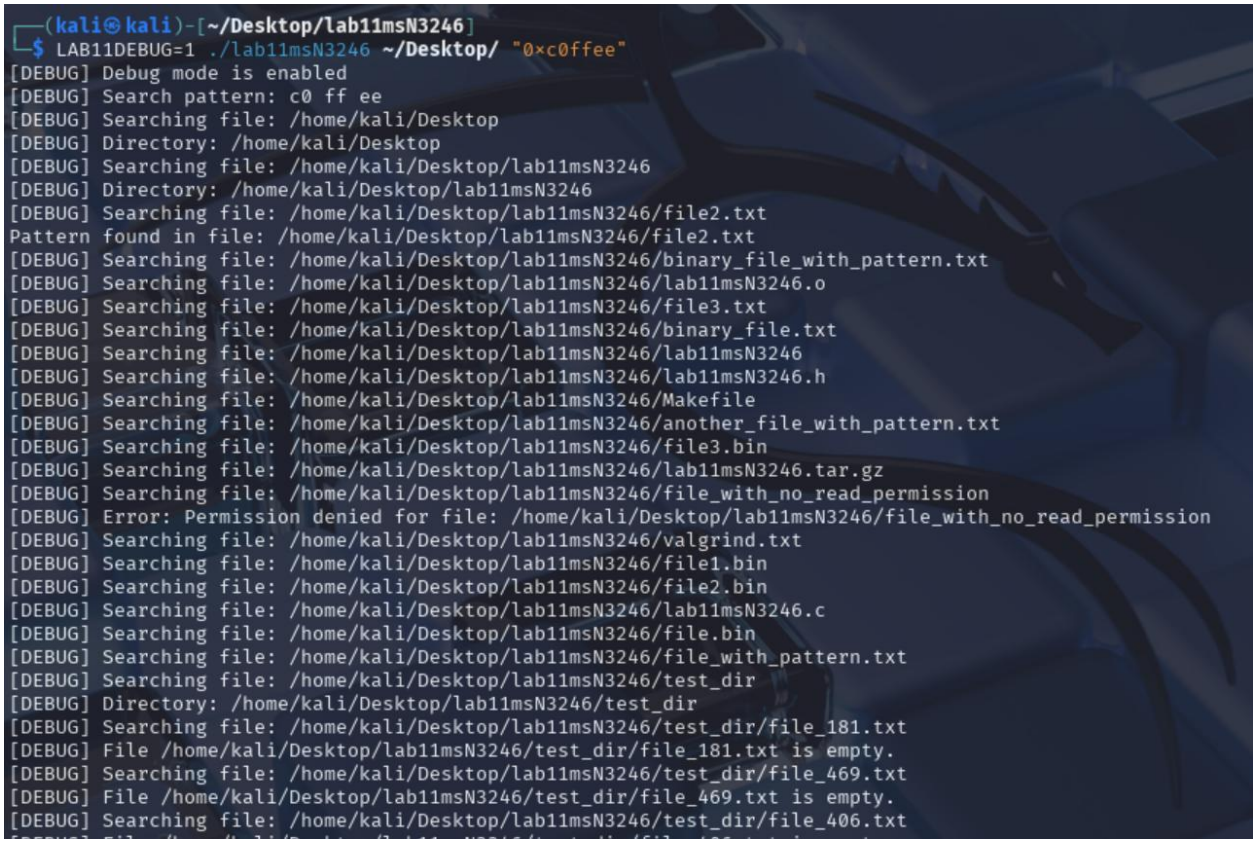

Рисунок 4 – Работа в отладочном режиме

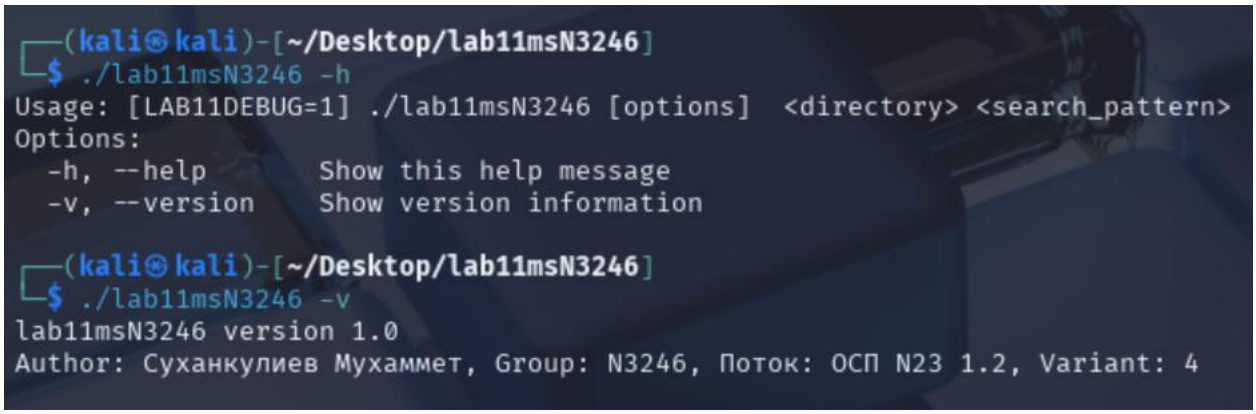

Рисунок 5 – Работа --help, --version

2.3 Тестирование утечек памяти

Команда `valgrind --leak-check=full … ./lab11msN3246 …` показала отсутствие утечек.

2.3.1 `valgrind.txt`

```
==7192== Memcheck, a memory error detector
==7192== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==7192== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==7192== Command: ./lab11msN3246 /usr/share/doc 0x68656c6c6f
==7192== Parent PID: 1800
==7192==
==7192==
==7192== HEAP SUMMARY:
==7192==     in use at exit: 0 bytes in 0 blocks
==7192==   total heap usage: 46,529 allocs, 46,529 frees, 235,410,157 bytes
allocated
==7192==
==7192== All heap blocks were freed -- no leaks are possible
==7192==
==7192== For lists of detected and suppressed errors, rerun with: -s
==7192== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

2.4 Выводы по тестированию

Программа корректно выполняет поиск, обрабатывает ошибки и не имеет утечек памяти.

ЗАКЛЮЧЕНИЕ

В ходе работы была разработана консольная утилита для рекурсивного поиска файлов, содержащих заданную последовательность байтов, с использованием функции `nftw()`. Программа обеспечивает корректную обработку ошибок, поддержку отладочного режима и сборку через `Makefile`. Тестирование показало стабильную работу программы, отсутствие утечек памяти (проверено через `valgrind`).

Это позволило закрепить навыки работы с системным программированием, обработкой файлов и каталогов, а также применением инструментов для тестирования и оптимизации программ (`-O3`).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган, Брайан У., Ритчи, Деннис М.: Язык программирования С, 2-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс» – 2009. – 304 с.: ил. – Парал. тит. англ. – URL: Керниган, Ритчи. Язык программирования С.pdf - Google Диск
2. Гирик А. В.: Основы системного программирования. Язык С. Сборка и отладка программ. Системные вызовы. Обработка ошибок. – Университет ИТМО – 2025. – URL: 01. ОСП. Язык С. Сборка и отладка программ. Системные вызовы. Обработка ошибок.pdf - Google Диск
3. Гирик А. В.: Основы системного программирования. Файлы и каталоги. Управление памятью – Университет ИТМО – 2025. – URL: 02. ОСП. Файлы и каталоги. Управление памятью.pdf - Google Диск

ПРИЛОЖЕНИЕ А

(обязательное)

Исходные коды программы с комментариями

Листинг А.1 – `lab11msN3246.h`

```
#ifndef LAB11MSN3246_H
#define LAB11MSN3246_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <errno.h>
#include <ftw.h>
#include <syslog.h>

#define MAX_PATTERN_SIZE 256
#define BUFFER_SIZE 4096

unsigned char *search_pattern = NULL;  // Динамическая память для шаблона
поиска
size_t pattern_size = 0;
int debug_mode = 0;  // Глобальная переменная для режима отладки

void print_help() {
    printf("Usage: [LAB11DEBUG=1] ./lab11msN3246 [options]  <directory>
<search_pattern>\n");
    printf("Options:\n");
    printf("  -h, --help       Show this help message\n");
    printf("  -v, --version    Show version information\n");
}

void print_version() {
    printf("lab11msN3246 version 1.0\n");
    printf("Author: Суханкулиев Мухаммет, Group: N3246, Поток: ОСП N23 1.2,
Variant: 4\n");
}

int is_hex_digit(char c) {
    return (c >= '0' && c <= '9') || (c >= 'a' && c <= 'f') || (c >= 'A' && c
<= 'F');
}

int parse_hex_pattern(const char *str) {
    if (strlen(str) < 4 || strncmp(str, "0x", 2) != 0) {
        fprintf(stderr, "Error: Search pattern must start with '0x' and be at
least 2 characters long after '0x'.\n");
        return -1;
    }

    size_t len = strlen(str) - 2;
    if (len % 2 != 0 || len / 2 > MAX_PATTERN_SIZE) {
        fprintf(stderr, "Error: Invalid search pattern length.\n");
        return -1;
    }

    for (size_t i = 2; i < 2 + len; i++) {
        if (!is_hex_digit(str[i])) {
```

```
            fprintf(stderr, "Error: Invalid hex format in search
pattern.\n");
            return -1;
        }
    }

    pattern_size = len / 2;
    search_pattern = (unsigned char *)malloc(pattern_size);
    if (!search_pattern) {
        fprintf(stderr, "Error: Memory allocation failed for search
pattern.\n");
        return -1;
    }

    for (size_t i = 0; i < pattern_size; i++) {
        sscanf(str + 2 + (i * 2), "%2hhx", &search_pattern[i]);
    }

    return 0;
}

int search_file(const char *path, const struct stat *statbuf, int type,
struct FTW *ftwbuf) {
    if (path == NULL || statbuf == NULL || ftwbuf == NULL) {
        fprintf(stderr, "[DEBUG] Error: NULL pointer passed to
search_file.\n");
        return 1;
    }

    if (debug_mode) {
        fprintf(stderr, "[DEBUG] Searching file: %s\n", path);
    }

    if (type == FTW_F) {
        FILE *file = fopen(path, "rb");  // Открываем файл для бинарного
чтения
        if (!file) {
            if (errno == EACCES) {
                if (debug_mode) {
                    fprintf(stderr, "Error: Permission denied for file:
%s\n", path);
                }
                return 0;  // Пропуск файла
            }
            perror("Error opening file");
            return 1;
        }

        size_t file_size = statbuf->st_size;
        if (file_size == 0) {
            if (debug_mode) {
                fprintf(stderr, "[DEBUG] File %s is empty.\n", path);
            }
            fclose(file);
            return 0;
        }

        unsigned char buffer[BUFFER_SIZE];
        size_t bytes_read = 0;
        int found = 0;

        while ((bytes_read = fread(buffer, 1, BUFFER_SIZE, file)) > 0) {
```

```
            for (size_t i = 0; i + pattern_size <= bytes_read; i++) {
                if (memcmp(buffer + i, search_pattern, pattern_size) == 0) {
                    printf("Pattern found in file: %s\n", path);
                    found = 1;
                    break;
                }
            }
            if (found) break;
        }

        if (bytes_read < BUFFER_SIZE && ferror(file)) {
            perror("Error reading file");
            fclose(file);
            return 1;
        }

        fclose(file);
    } else if (type == FTW_D) {
        if (debug_mode) {
            fprintf(stderr, "[DEBUG] Directory: %s\n", path);
        }
    } else {
        if (debug_mode) {
            fprintf(stderr, "[DEBUG] Other type: %s\n", path);
        }
    }

    return 0;
}

void log_error(const char *msg) {
    openlog("lab11msN3246", LOG_PID | LOG_CONS, LOG_USER);
    syslog(LOG_ERR, "%s", msg);
    closelog();
}

#endif // LAB11MSN3246_H
```

Листинг А.2 – `lab11msN3246.c`

```
#define _XOPEN_SOURCE 700  // Для nftw()
#include "lab11msN3246.h"

int main(int argc, char *argv[]) {
    int opt;
    int show_help = 0, show_version = 0;

    struct option long_options[] = {
        {"help", no_argument, 0, 'h'},
        {"version", no_argument, 0, 'v'},
        {0, 0, 0, 0}
    };

    // Проверка LAB11DEBUG
    debug_mode = (getenv("LAB11DEBUG") != NULL);
    if (debug_mode) {
        fprintf(stderr, "[DEBUG] Debug mode is enabled\n");
    }

    // Обработка опций командной строки
    while ((opt = getopt_long(argc, argv, "hv", long_options, NULL)) != -1) {
```

```
        switch (opt) {
            case 'h':
                show_help = 1;
                break;
            case 'v':
                show_version = 1;
                break;
            default:
                fprintf(stderr, "Unknown option. Use -h for help.\n");
                return 1;
        }
    }
    if (show_help) {
        print_help();
    }
    if (show_version) {
        print_version();
    }
    if (show_help || show_version) {
        return 0;
    }

    // Доп. проверка аргументов
    if (optind + 2 != argc) {
        fprintf(stderr, "Usage: %s [options] <directory> <search_pattern>\n",
argv[0]);
        return 1;
    }

    // Парсинг шаблона поиска
    if (parse_hex_pattern(argv[optind + 1]) != 0) {
        return 1;
    }
    if (debug_mode) {
        fprintf(stderr, "[DEBUG] Search pattern: ");
        for (size_t i = 0; i < pattern_size; i++) {
            fprintf(stderr, "%02x ", search_pattern[i]);
        }
        fprintf(stderr, "\n");
    }

    // Обработка каталога и выполнение поиска
    const char *dir = argv[optind];
    if (nftw(dir, search_file, 20, FTW_PHYS) == -1) {
        perror("Error processing directory");
        log_error("Error processing directory.");
        return 1;
    }
    free(search_pattern);
    return 0;
}
```