

Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Операционные системы и системное программирование

## ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

по теме

ЗНАКОМСТВО С LINUX/UNIX И СРЕДОЙ ПРОГРАММИРОВАНИЯ  
POSIX-СОВМЕСТИМАЯ ФАЙЛОВАЯ СИСТЕМА

БГУИР КР 1-40 02 01 001 ЛР

Выполнил: студент группы 250541,  
Бобрик В. Ю.

Проверил: ст.преподаватель каф. ЭВМ,  
Поденок Л. П.

Минск 2025

## СОДЕРЖАНИЕ

|   |   |
|---|---|
| 1 Условия лабораторной работы.....        | 3 |
| 2 Описание алгоритмов и решений.....      | 4 |
| 3 Функциональная структура проекта.....   | 5 |
| 4 Порядок сборки и тестирования.....      | 6 |
| 5 Методика и результаты тестирования..... | 9 |

## 1 Условия лабораторной работы

Изучение оболочки `bash`, файлового менеджера `mc`, обеспечение: `linux`, стандартное информационное обеспечение (`info`, `man`).

Внешнее знакомство с POSIX-совместимой файловой системой  
структура каталогов, – жесткие и символические ссылки, права доступа, монтирование файловых систем, монтирование каталогов (`mount`, `mount – bind`).

Изучаемые вопросы:

- команды и утилиты: `man`, `info`, `mkdir`, `touch`, `rm`, `rmdir`, `cd`, `cat`, `sort`, `head`, `tail`, `tee`, `wc`, `chmod`, `ls`, `lsof`, `lsblk`, `lsusb`, `lscpu`, `ln`, `link`, `unlink`, `locale`, `iconv`, `kill`, `top`, `htop`, `ps`, `grep`, `diff`, `env`, `file`, `stat`, `find`, `tar`, `gzip`, `more`, `less`, `printf`, `time`, ...;

- сцепление программ и соединение выходных и входных стандартных потоков;

- перенаправление вывода `stdout` и `stderr` в файлы;

- структура ФС, содержимое `inode`, команды оболочки;

- знакомство с POSIX-совместимой файловой системой – `opendir(3)`, `readdir(3)`, `closedir(3)`, `stat(2)`, `lstat(2)`, `readlink(2)`, `realpath(1)`, `symlink(2)`, `link(2)`, `unlink(2)`, ...

### Задание

Освоить эффективную работу с файлами в оболочке и `mc`.

Разработать программу `dirwalk`, сканирующую файловую систему и выводящую в `stdout` информацию в соответствии с опциями программы.

Формат вывода аналогичен формату вывода утилиты `find`.

`dirwalk [dir] [options]`

`dir` – начальный каталог. Если опущен, текущий (`./`).

`options` – опции.

`-l` – только символические ссылки (`-type l`)

`-d` – только каталоги (`-type d`)

`-f` – только файлы (`-type f`)

`-s` – сортировать выход в соответствии с `LC_COLLATE`

Опции могут быть указаны как перед каталогом, так и после.

Опции могут быть указаны как отдельно, так и вместе (`-l -d`, `-ld`).

Если опции `ldf` опущены, выводятся каталоги, файлы и ссылки.

Для обработки опций рекомендуется использовать `getopt(3)`.

Программа должна быть переносимой (возможности `linux` не используются, только `posix`).

## 2 Описание алгоритмов и решений

Программа `dirwalk` выполняет обход файловой системы, анализирует содержимое каталогов и фильтрует результаты согласно переданным опциям.

Алгоритм анализа аргументов командной строки имеет целью определение каталога и набора опций.

Шаг 1. Инициализация структуры `options_s` с установленными флагами `-l`, `-d`, `-f`, `-s` в значение 0.

Шаг 2. Использование `getopt(3)` для чтения переданных флагов.

Шаг 3. Проверка `optind`: если после опций есть аргумент, он интерпретируется как начальный каталог (`start_dir`). Если его нет, используем каталог `./` по умолчанию.

Шаг 4. Установка флагов согласно аргументам.

Программа `dirwalk` использует алгоритм рекурсивного глубинного обхода. Алгоритм посещает все файлы и подкаталоги, анализирует их тип и фильтрует согласно параметрам запуска.

Шаг 1. Открыть переданный каталог (`opendir( )`).

Шаг 2. Прочитать содержимое каталога (`readdir( )`).

Шаг 3. Игнорировать . и .., чтобы не заикливать программу на текущем и родительском каталоге.

Шаг 4. Сформировать абсолютный или относительный путь к объекту.

Шаг 5. Получить тип файла (lstat( )).

Шаг 6. Сравнить тип файла с переданными флагами (-l -d -f).  
S\_ISREG(sb.st\_mode) - обычный файл. S\_ISDIR(sb.st\_mode) - каталог.  
S\_ISLNK(sb.st\_mode) - символическая ссылка.

Шаг 7. Если объект – каталог (S\_ISDIR), рекурсивно вызвать dirwalk.

Шаг 8. Сохранить путь к объекту в массив.

Шаг 9. Вывести массив путей в stdout.

### **3 Функциональная структура проекта**

Проект состоит из нескольких модулей.

Модуль обработки командной строки (options) отвечает за разбор аргументов, переданных в командной строке, и выделение из них двух основных типов данных.

Структура options: содержит флаги для каждого режима вывода.

```
typedef struct {  
    int type_links; // Флаг для опции -l (символические ссылки)  
    int type_dirs;  // Флаг для опции -d (только каталоги)  
    int type_files; // Флаг для опции -f (только обычные файлы)  
    int sort;       // Флаг для опции -s (сортировка LC_COLLATE)  
} options;
```

Функция parse\_options( ): использует POSIX-функцию getopt(3) для анализа аргументов и заполнения структуры Options.

Модуль рекурсивного обхода файловой системы (dirwalk) обходит все файлы и подкаталоги, начиная с заданного каталога, с последующей фильтрацией и при необходимости сортировкой полученного списка.

Структура `filelist` используется для хранения списка найденных путей.

```
typedef struct {  
    char **items;  
    size_t size;  
    size_t capacity;  
} filelist;
```

`dirwalk()` – начальная точка для рекурсивного обхода, которая принимает стартовый каталог и структуру опций.

`dirwalk_internal()` – рекурсивная функция, которая обрабатывает отдельный каталог: формирует пути, фильтрует их и, если объект — каталог, вызывает саму себя.

`match_type()` – функция, проверяющая соответствие объекта заданным флагам.

`create_file_list()` – функция, создающая список `filelist`.

`add_to_list()` – функция добавления объекта в `filelist`.

`free_file_list()` – очистка `filelist` по окончании работы программы.

`print_file_list()` – вывод списка объектов `filelist` в `stdout`.

Точка входа (`main`) – главный модуль программы объединяет отдельные модули и определяет порядок выполнения. Содержит функцию `main()`, определяющую алгоритм работы программы.

## 4 Порядок сборки и тестирования

Для сборки используется `Makefile`, содержащий следующие ключевые элементы:

Исходные файлы находятся в каталоге `./src`.

Заголовочные файлы расположены в каталоге `./include`.

В зависимости от типа сборки создаётся каталог `./out/debug` для debug-версии или `./out/release` для release-версии.

Переменная BUILD задаётся как debug по умолчанию. При запуске make release устанавливается оптимизированная сборка.

Флаги компиляции задаются через переменные COMMON\_CFLAGS (определения, предупреждения, стандарт C11) и дополняются флагами отладки (-g -O0) или оптимизации (-O2).

Для сборки в debug-режиме необходимо выполнить в терминале:  
make

Это соберёт программу в ./out/debug/dirwalk.

Для сборки в release-режиме необходимо выполнить в терминале:  
make release

Это соберёт программу в ./out/release/dirwalk.

Для очистки сборки необходимо выполнить в терминале:  
make clean

Для тестирования используется папка test, в которой расположена следующая структура:

```
test/
├─ Ruby74
├─ ┬─ ruby-file.txt
├─ test_debug.sh
├─ test_release.sh
├─ Wallet13
├─ ┬─ Charlie13-17
├─ │ ┬─ Charlie-01.txt
├─ │ ┬─ Charlie-02.txt
├─ │ └─ Charlie-03.txt
├─ ruby-link -> test/Ruby74/ruby-file.txt
└─ Wallet-01.txt
```

Для проверки работы программы можно использовать различные комбинации опций. Например:

1. Запуск без опций:

Выполнить:

./out/release/dirwalk test/

Программа выведет все найденные файлы и каталоги, начиная с каталога test/.

```
[silvarious@fedora lab01]$ out/release/dirwalk test/
test/Ruby74
test/Ruby74/ruby-file.txt
test/Wallet13
test/Wallet13/Charlie13-17
test/Wallet13/Charlie13-17/Charlie-01.txt
test/Wallet13/Charlie13-17/Charlie-02.txt
test/Wallet13/Charlie13-17/Charlie-03.txt
test/Wallet13/Wallet-01.txt
test/Wallet13/ruby-link
test/test_debug.sh
test/test_release.sh
```

## 2. Фильтрация по файлам (-f):

Для вывода только файлов:

```
./out/debug/dirwalk test/ -f
```

Ожидается, что будут выведены пути типа test/.../\*.txt без каталогов.

```
[silvarious@fedora lab01]$ out/release/dirwalk test/ -f
test/Ruby74/ruby-file.txt
test/Wallet13/Charlie13-17/Charlie-01.txt
test/Wallet13/Charlie13-17/Charlie-02.txt
test/Wallet13/Charlie13-17/Charlie-03.txt
test/Wallet13/Wallet-01.txt
test/test_debug.sh
test/test_release.sh
```

## 3. Вывод только каталогов (-d):

Выполнить:

```
./out/debug/dirwalk test/ -d
```

Программа покажет только каталоги из структуры test/.

```
[silvarious@fedora lab01]$ out/release/dirwalk test/ -d
test/Ruby74
test/Wallet13
test/Wallet13/Charlie13-17
```

## 4. Вывод только символических ссылок (-l):

При условии, что тестовые символические ссылки созданы, выполнить:

```
./out/debug/dirwalk test/ -l
```



Ожидается вывод только ссылок.

```
[silvarious@fedora lab01]$ out/release/dirwalk test/ -l  
test/Wallet13/ruby-link
```

5. Сортировка результата (-s):

Выполнить:

```
./out/debug/dirwalk test/ -s
```

Ожидается вывод в алфавитном порядке согласно LC\_COLLATE.

```
[silvarious@fedora lab01]$ out/release/dirwalk test/ -s  
test/Ruby74  
test/Ruby74/ruby-file.txt  
test/test_debug.sh  
test/test_release.sh  
test/Wallet13  
test/Wallet13/Charlie13-17  
test/Wallet13/Charlie13-17/Charlie-01.txt  
test/Wallet13/Charlie13-17/Charlie-02.txt  
test/Wallet13/Charlie13-17/Charlie-03.txt  
test/Wallet13/ruby-link  
test/Wallet13/Wallet-01.txt
```

## 5 Методика и результаты тестирования

Чтобы не запускать каждую команду вручную, созданы bash-скрипты test\_debug.sh и test\_release.sh, которые выполняют последовательные тестовые сценарии.

```
#!/bin/bash  
# Скрипт для тестирования программы dirwalk  
clear  
echo "Тест 1: Проверка без каких-либо опций"  
out/release/dirwalk test/  
echo ""  
echo "Тест 2: Вывод только файлов (-f)"  
out/release/dirwalk test/ -f  
echo ""  
echo "Тест 3: Вывод только каталогов (-d)"  
out/release/dirwalk test/ -d  
echo ""  
echo "Тест 4: Вывод только символических ссылок (-l)"  
out/release/dirwalk test/ -l  
echo ""  
echo "Тест 5: Сортировка вывода (-s)"  
out/release/dirwalk test/ -s
```

```
echo ""
echo "Тест 6: Комбинированный запуск (-f -l -s)"
out/release/dirwalk test/ -f -l -s
echo ""
echo "Тестирование завершено."
```

Скрипт необходимо назначить исполняемым и запустить:

```
chmod +x test_*.sh
./test/test_release.sh
```

Результат автоматического тестирования:

Тест 1: Проверка без каких-либо опций

```
test/Ruby74
test/Ruby74/ruby-file.txt
test/Wallet13
test/Wallet13/Charlie13-17
test/Wallet13/Charlie13-17/Charlie-01.txt
test/Wallet13/Charlie13-17/Charlie-02.txt
test/Wallet13/Charlie13-17/Charlie-03.txt
test/Wallet13/Wallet-01.txt
test/Wallet13/ruby-link
test/test_debug.sh
test/test_release.sh
```

Тест 2: Вывод только файлов (-f)

```
test/Ruby74/ruby-file.txt
test/Wallet13/Charlie13-17/Charlie-01.txt
test/Wallet13/Charlie13-17/Charlie-02.txt
test/Wallet13/Charlie13-17/Charlie-03.txt
test/Wallet13/Wallet-01.txt
test/test_debug.sh
test/test_release.sh
```

Тест 3: Вывод только каталогов (-d)

```
test/Ruby74
test/Wallet13
test/Wallet13/Charlie13-17
```

Тест 4: Вывод только символических ссылок (-l)

```
test/Wallet13/ruby-link
```

Тест 5: Сортировка вывода (-s)

```
test/Ruby74
test/Ruby74/ruby-file.txt
test/test_debug.sh
test/test_release.sh
test/Wallet13
```

```
test/Wallet13/Charlie13-17
test/Wallet13/Charlie13-17/Charlie-01.txt
test/Wallet13/Charlie13-17/Charlie-02.txt
test/Wallet13/Charlie13-17/Charlie-03.txt
test/Wallet13/ruby-link
test/Wallet13/Wallet-01.txt
```

```
Тест 6: Комбинированный запуск (-f -l -s)
test/Ruby74/ruby-file.txt
test/test_debug.sh
test/test_release.sh
test/Wallet13/Charlie13-17/Charlie-01.txt
test/Wallet13/Charlie13-17/Charlie-02.txt
test/Wallet13/Charlie13-17/Charlie-03.txt
test/Wallet13/ruby-link
test/Wallet13/Wallet-01.txt
```

Тестирование завершено.