Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра ПОИТ

Дисциплина «Архитектура компьютерной техники и операционных систем»

ОТЧЁТ
к лабораторной работе №6

РАБОТА С ФАЙЛАМИ И КАТАЛОГАМИ ОС LINUX

Вариант 8

Студент группы №351001
Ушаков А.Д.
Преподаватель
Леванцевич В.А.

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Операционные системы (ОС) играют ключевую роль в управлении компьютерными ресурсами и обеспечении взаимодействия между аппаратным обеспечением и пользователями. В этом отчёте рассматриваются основные аспекты, касающиеся архитектуры и организации ОС, а также механизмы работы с файловыми системами. Понимание уровней привилегий, режимов работы и системных вызовов является необходимым для глубокой оценки функциональности и безопасности современных операционных систем.

Уровни привилегий в ОС делятся на два режима: пользовательский и ядровой. Пользовательский режим ограничивает доступ к критическим ресурсам системы, обеспечивая защиту от неправомерного использования. Ядровой режим, наоборот, предоставляет полный доступ к аппаратным ресурсам и системным вызовам. Системные вызовы служат интерфейсом между пользовательскими приложениями и ядром ОС, позволяя программам запрашивать выполнение различных операций, таких как работа с файлами или управление процессами. В Unix/Linux эта реализация происходит через специальные функции, которые обрабатываются ядром.

Архитектура операционных систем может быть представлена в виде монолитной или микроядерной структуры. Монолитные ядра объединяют все основные функции в одном модуле, что обеспечивает высокую производительность, но может привести к сложности в разработке и отладке. Микроядерные архитектуры, напротив, разделяют функции на отдельные модули, что облегчает их обновление и тестирование, но может увеличить накладные расходы на взаимодействие между компонентами.

Структура ОС Linux и её ядра отражает гибкость и модульность, позволяя пользователям настраивать систему под свои потребности. Важно также учитывать различные типы пользователей, таких как администраторы и обычные пользователи, у которых разные уровни доступа и привилегий. Это различие определяет, как пользователи взаимодействуют с системой и какие операции они могут выполнять.

Работа с файлами в ОС организована через структуры данных, такие как DIR и dirent, которые обеспечивают доступ к содержимому каталогов. Формат структур работы с файлами, включая статические данные, такие как stat, позволяет эффективно управлять файлами и их атрибутами. Важным аспектом является организация файловых систем. Например, файловая система FAT использует связанный список блоков для управления данными, в то время как система i-nodes, используемая в ext, предоставляет более гибкий подход к хранению информации о файлах и их метаданных.

Монтирование файловой системы — это процесс интеграции файловой системы в иерархию каталогов, который позволяет пользователям получать доступ к данным на различных носителях. Этот процесс критически важен для обеспечения единого пространства имен, через которое пользователи и приложения могут взаимодействовать с файлами.

Кроме того, понимание структуры и организации файловых систем помогает разработчикам и системным администраторам оптимизировать производительность и управлять ресурсами более эффективно. Знание о том, как различные файловые системы взаимодействуют с операционной системой, позволяет лучше проектировать приложения и системы хранения данных, адаптируя их под специфические требования пользователей и задач.

Исследование современных операционных систем и их архитектуры предоставляет ценные знания как для профессионалов в области информационных технологий, так и для студентов, изучающих эту область. Глубокое понимание принципов работы ОС и файловых систем способствует разработке более надежных и эффективных программных решений, что, в свою очередь, влияет на общий уровень производительности и безопасности вычислительных систем.

1 ЗАДАНИЕ К РАБОТЕ

1.1 Требования к выполнению

Написать программу на языке С, согласно варианту задания.

Минимальные требования к отчету:

1 Отчет оформляется в бумажном виде.

2 Отчет должен содержать титульный лист с фамилией и номером варианта.

3 В отчете должны присутствовать: текст задания и листинг программы.

Варианты заданий выбираются по списку подгруппы. Первые девять человек выбирают номера 1-9, вторые девять человек также выбирают номера 1-9 и т.д.

1.2 Индивидуальное задание

К варианту 8 относится задание: написать программу, находящую в заданном каталоге (первый аргумент командной строки) и всех его подкаталогах все файлы заданного расширения и создающую для каждого найденного файла жесткую ссылку в заданном каталоге. Расширение файла и каталог для жестких ссылок задаются в качестве второго и третьего аргументов командной строки.

Представьте, что дерево на рисунке 1.2.1 – файловая система Linux, и изучать результаты лабораторной работы станет намного легче!

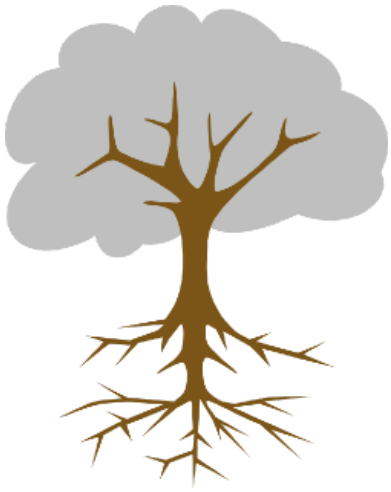

Рисунок 1.2.1 – Файловая система Linux (карикатура)

2 РЕАЛИЗАЦИЯ ПРОГРАММЫ

2.1 Код программы

```
#ifndef PATH_MAX
#define PATH_MAX 4096  /* Определение PATH_MAX, если он не задан */
#endif

#include <stdio.h>      /* Подключение стандартной библиотеки ввода-
вывода */
#include <stdlib.h>     /* Подключение стандартной библиотеки общего
назначения (например, для EXIT_FAILURE и EXIT_SUCCESS) */
#include <string.h>     /* Подключение библиотеки для работы со
строками (например, функции strcmp и strrchr) */
#include <sys/types.h>  /* Подключение типов данных системных вызовов
*/
#include <sys/stat.h>   /* Подключение библиотеки для работы с
информацией о файлах и каталогах */
#include <dirent.h>     /* Подключение библиотеки для работы с
каталогами */
#include <unistd.h>     /* Подключение POSIX-библиотеки (например, для
функции link) */
#include <libgen.h>     /* Подключение библиотеки для работы с именами
файлов и путями */
#include <limits.h>  /* Подключение для определения значения PATH_MAX
*/

/* Функция для рекурсивного поиска файлов в каталоге и создания
жестких ссылок */
void find_files(const char *dir_path, const char *extension, const
char *link_dir) {
    DIR *dir = opendir(dir_path);  /* Открытие каталога по указанному
пути */
    if (!dir) {  /* Проверка на успешное открытие каталога */
        perror("Ошибка открытия каталога");  /* Вывод сообщения об
ошибке, если каталог не открыт */
        return;  /* Выход из функции, если каталог не открыт */
    }

    struct dirent *entry;  /* Указатель на структуру, представляющую
запись в каталоге */
    while ((entry = readdir(dir)) != NULL) {  /* Чтение всех записей в
каталоге */
        /* Пропускаем "." и ".." */
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name,
"..") == 0)
            continue;

        /* Построение полного пути к текущему элементу */
```

```
        char full_path[PATH_MAX];  /* Буфер для полного пути к файлу
*/
        snprintf(full_path, sizeof(full_path), "%s/%s", dir_path,
entry->d_name);  /* Формирование полного пути */

        struct stat st;  /* Структура для хранения информации о файле
*/
        if (stat(full_path, &st) == -1) {  /* Получение информации о
файле */
            perror("Ошибка получения информации о файле");  /* Вывод
сообщения об ошибке при сбое stat */
            continue;  /* Переход к следующей записи в каталоге, если
не удалось получить информацию */
        }

        /* Если это каталог, рекурсивно обходим его */
        if (S_ISDIR(st.st_mode)) {  /* Проверка, является ли запись
каталогом */
            find_files(full_path, extension, link_dir);  /*
Рекурсивный вызов для обхода подкаталога */
        }
        /* Если это файл, проверяем его расширение */
        else if (S_ISREG(st.st_mode)) {  /* Проверка, является ли
запись обычным файлом */
            const char *ext = strrchr(entry->d_name, '.');  /* Поиск
последнего вхождения '.' в имени файла для проверки расширения */
            if (ext && strcmp(ext, extension) == 0) {  /* Сравнение
найденного расширения с заданным */
                /* Построение пути для жесткой ссылки */
                char link_path[PATH_MAX];  /* Буфер для пути к жесткой
ссылке */
                snprintf(link_path, sizeof(link_path), "%s/%s",
link_dir, entry->d_name);  /* Формирование полного пути для жесткой
ссылки */

                /* Проверка, существует ли уже жесткая ссылка */
                struct stat link_st;
                if (stat(link_path, &link_st) == 0) {  /* Если stat
возвращает 0, значит файл существует */
                    printf("Жесткая ссылка уже существует: %s\n",
link_path);  /* Сообщение о существующей ссылке */
                } else {
                    /* Создание жесткой ссылки */
                    if (link(full_path, link_path) == -1) {  /*
Попытка создать жесткую ссылку */
                        perror("Ошибка создания жесткой ссылки");  /*
Вывод сообщения об ошибке при сбое link */
                    } else {
                        printf("Жесткая ссылка создана: %s -> %s\n",
full_path, link_path);  /* Уведомление о созданной жесткой ссылке */
                    }
```

```
                }
            }
        }
    }

    closedir(dir);  /* Закрытие каталога после завершения обработки
всех записей */
}

int main(int argc, char *argv[]) {
    if (argc != 4) {  /* Проверка количества аргументов командной
строки */
        fprintf(stderr, "Использование: %s <каталог поиска>
<расширение> <каталог для ссылок>\n", argv[0]);  /* Вывод справки по
использованию программы */
        return EXIT_FAILURE;  /* Завершение программы с кодом ошибки,
если аргументы не совпадают */
    }

    /* Проверяем, существует ли каталог для ссылок */
    struct stat st;  /* Структура для хранения информации о каталоге
для ссылок */
    if (stat(argv[3], &st) == -1 || !S_ISDIR(st.st_mode)) {  /*
Проверка существования каталога для ссылок и его типа */
        fprintf(stderr, "Каталог для ссылок не существует или не
является каталогом: %s\n", argv[3]);  /* Вывод ошибки, если каталог не
существует или не является каталогом */
        return EXIT_FAILURE;  /* Завершение программы с кодом ошибки
*/
    }

    /* Запуск поиска файлов и создания жестких ссылок */
    find_files(argv[1], argv[2], argv[3]);  /* Вызов функции поиска
файлов и создания ссылок */

    return EXIT_SUCCESS;  /* Завершение программы с успешным кодом */
}
```

2.2 Запуск исполняемого файла

Для запуска программы необходимо перейти в директорию /home/sashka/Projects/ЛР6. После этого нужно запустить исполняемый файл с помощью команды `./task /home/sashka/Projects/files .txt /home/sashka/Projects/ЛР6/Links` (см. рисунок 2.2.1), где первый аргумент – сам исполняемый файл, второй – проверяемая директория, третий – интересующее расширение файла, четвёртый – путь к каталогу, в котром будут создаваться жёсткие ссылки.

```
sashka@sashka-VMware-Virtual-Platform:~$ cd /home/sashka/Projects/ЛР6
sashka@sashka-VMware-Virtual-Platform:~/Projects/ЛР6$ ./task /home/sashka/Projec
ts/files .txt /home/sashka/Projects/ЛР6/Links
Жесткая ссылка создана: /home/sashka/Projects/files/files/4.txt -> /home/sashka/
Projects/ЛР6/Links/4.txt
Жесткая ссылка создана: /home/sashka/Projects/files/files/5.txt -> /home/sashka/
Projects/ЛР6/Links/5.txt
Жесткая ссылка создана: /home/sashka/Projects/files/1.txt -> /home/sashka/Projec
ts/ЛР6/Links/1.txt
Жесткая ссылка создана: /home/sashka/Projects/files/2.txt -> /home/sashka/Projec
ts/ЛР6/Links/2.txt
Жесткая ссылка создана: /home/sashka/Projects/files/3.txt -> /home/sashka/Projec
ts/ЛР6/Links/3.txt
```

Рисунок 2.2.1 – Запуск исполняемого файла

2.3 Результат работы программы

На рисунке 2.3.1 продемонстрировано, как в папке Links создаются жёсткие ссылки на текстовые документы, находящиеся в каталоге files и его подкаталогах.

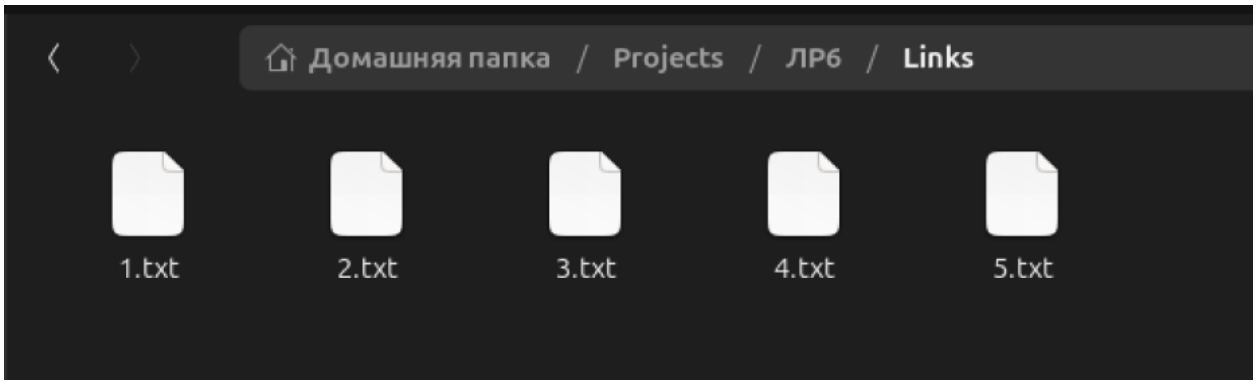

Рисунок 2.3.1 – Результат работы программы

ЗАКЛЮЧЕНИЕ

В процессе работы были изучены ключевые функции и системные вызовы ОС Linux, обеспечивающие взаимодействие с файлами и каталогами. Рассматривались функции для открытия и создания файлов (`open()` и `fopen()`), а также параметры доступа, регулируемые флагами, что позволило понять, как задавать и проверять права на чтение, запись и выполнение.

Существенное внимание уделено структурам `stat` и `dirent`, которые позволяют получать расширенную информацию о файлах и каталогах, такую как размер файла, время его последнего изменения, права доступа и номер индексного дескриптора. Изучение этих структур дало возможность реализовать эффективное считывание характеристик файлов и каталогов в автоматическом режиме.

Для работы с каталогами были применены функции `opendir()`, `readdir()`, и `closedir()`, позволяющие считывать данные о содержимом каталогов, а также выполнять навигацию по файловой структуре. Эти вызовы обеспечили удобный способ получения списка файлов и подкаталогов и стали основой для анализа и обработки файловой системы.

Также изучены способы управления правами доступа с помощью `chmod()`, позволяющего менять доступ к файлам и каталогам, а функции `mkdir()` и `rmdir()` дали возможность создавать и удалять каталоги. Этот функционал полезен для контроля над файловой структурой и доступа к ней.

Выполнение индивидуального задания позволило применить на практике системные вызовы и функции, требуемые для манипуляций с файлами и каталогами. Работа над заданием помогла лучше понять, как на низком уровне ОС Linux предоставляет доступ к данным в файловой системе, и как их можно использовать для анализа и управления содержимым каталогов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Леванцевич, В. А. Работа с файлами и каталогами ОС Linux / В. А. Леванцевич // Лабораторные работы. – 2024. – №6.