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

ПОЯСНИТЕЛЬНАЯ
ЗАПИСКА к курсовому проекту
на тему

СИСТЕМНЫЙ ИНТЕРФЕЙС UFS-ПОДОБНОЙ ФАЙЛОВОЙ СИСТЕМЫ (LINUX)

БГУИР КП 1-40 02 01 412 ПЗ

Студент:
группы 150504,
Лужков И.А.

Руководитель:
Басак Д.В.

Минск 2023

СОДЕРЖАНИЕ

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В данной курсовой работе будет рассмотрена тема "Системный интерфейс UFS-подобной файловой системы (Linux)" с целью изучения принципов и основ проектирования подобных систем.

На сегодняшний день, файловые системы играют важную роль в операционных системах, обеспечивая хранение, организацию и управление файлами. Однако, различные задачи требуют различных типов файловых систем, что в свою очередь требует разработки новых систем или изменений уже существующих. В рамках данного дипломного проекта рассматривается UFS-подобная файловая система для Linux.

Целью дипломного проектирования является разработка системного интерфейса для UFS-подобных файловых систем на платформе Linux, который обеспечивает эффективное взаимодействие между системой и файловыми структурами.

Проектирование системного интерфейса исходит из базовых принципов функционирования UFS-подобных файловых систем и определенных научных исследований, проведенных в этой области. При этом особое внимание уделяется поиску технических решений для обеспечения высокой производительности и надежности файловой системы.

В пояснительной записке курсовой работы будут рассмотрены следующие разделы:

- Обзор UFS-подобных файловых систем
- Анализ существующих файловых систем в Linux
- Проектирование системного интерфейса
- Реализация UFS-подобной файловой системы на Linux
- Тестирование и анализ результатов Каждый из этих разделов будет посвящен решению конкретных задач, направленных на достижение цели исследования.

1. ОБЗОР ЛИТЕРАТУРЫ

1.1 • История развития файловых систем в Linux

История развития файловых систем в Linux началась с появления первых версий ядра Linux в 1991 году. Оригинальные версии ядра Linux использовали файловую систему Minix, но были быстро модифицированы, чтобы поддерживать другие типы файловых систем.

В 1993 году была разработана файловая система ext, которая заменила Minix как стандартная файловая система для Linux. Файловая система ext была разработана с учетом специфических требований операционной системы Linux и предоставляла новые возможности, такие как поддержка файлов с размером более 2 Гб и более эффективное управление свободным пространством на диске.

В последующие годы было создано несколько новых файловых систем, таких как XFS, JFS, ReiserFS и другие. Каждая из этих файловых систем имела свои особенности и предназначалась для определенных задач. Например, XFS была разработана для эффективной работы с большими файлами и объемами данных, а JFS - для обеспечения надежности и безопасности данных.

С появлением все большего количества цифровых устройств, таких как флеш-накопители, карты памяти и т.д., возникла потребность в разработке файловых систем для них. В результате появилась файловая система FAT, которая была создана для использования на съемных устройствах, таких как USB-накопители.

В последние годы были разработаны новые файловые системы, такие как btrfs и ZFS, которые предоставляют новые возможности, такие как журналирование метаданных, копирование на запись и снапшоты. Btrfs была разработана для использования на больших массивах данных и обеспечивает эффективное резервное копирование и восстановление данных. ZFS, в свою очередь, является файловой системой с высокой степенью отказоустойчивости и безопасности, которая поддерживает снапшоты и копирование на запись.

Таким образом, развитие файловых систем в Linux продолжается и постоянно появляются новые решения и возможности. Каждая из файловых систем имеет свои особенности и предназначена для определенной задачи, и выбор конкретной файловой системы зависит от требований и задачи, которую необходимо выполнить.

Одним из направлений развития файловых систем в Linux является улучшение производительности и эффективности работы с данными. Например, были разработаны файловые системы F2FS и NILFS2, которые оптимизированы для работы с флеш-накопителями и обеспечивают высокую скорость записи и чтения данных.

Другим важным направлением развития файловых систем является защита данных от потери и повреждения. Для этого были созданы файловые системы с механизмами резервного копирования и восстановления данных, такие как btrfs и ZFS.

Кроме того, появилась потребность в использовании файловых систем в распределенных средах, например, в кластерах серверов или облачных хранилищах. В связи с этим были разработаны новые файловые системы, такие как GlusterFS и Ceph, которые обеспечивают распределенную работу с данными на нескольких узлах.

В целом, можно отметить, что развитие файловых систем в Linux является динамичным процессом, который продолжается и совершенствуется по мере появления новых технологий и потребностей пользователей. Важно понимать, что выбор определенной файловой системы зависит от конкретных требований и задачи, которую необходимо решить. Поэтому важно знать особенности каждой из файловых систем и уметь выбирать наиболее подходящее решение.

1.2 Обзор UFS-подобных файловых систем

UFS (Unix File System) - это стандартная файловая система для операционных систем семейства Unix, она была разработана в 1970-х годах и

продолжает использоваться на многих Unix-подобных ОС до сих пор. UFS-подобные файловые системы используют подобные концепции и алгоритмы управления файлами, что делает их более понятными для пользователей и администраторов, знакомых с Unix.

Одной из особенностей UFS-подобных файловых систем является использование индексированного узла дерева (I-узел), который хранит информацию о каждом файле и директории в файловой системе. Каждый I-узел содержит метаданные файла, такие как права доступа, время создания и изменения, размер файла и т.д. Благодаря этому каждый файл можно быстро найти и получить доступ к его метаданным.

В Linux существует несколько UFS-подобных файловых систем, таких как FFS (Fast File System) и Ext2 (Extended Filesystem 2). FFS была первой файловой системой, которую использовали в Unix, и до сих пор она широко используется на серверах Unix. Ext2 была разработана в 1993 году как более эффективная и надежная альтернатива FFS. Она была первой файловой системой, которая стала стандартной для Linux и остается популярной до сих пор.

Ext2 имеет множество преимуществ по сравнению с FFS, таких как более быстрое создание, удаление и перемещение файлов, поддержка больших дисковых разделов и файлов, а также возможность отмены последних операций. В последующие годы были разработаны новые версии Ext2, такие как Ext3 и Ext4, которые добавили еще больше функциональности и оптимизировали производительность работы с данными.

Кроме того, существуют другие UFS-подобные файловые системы, такие как HFS (Hierarchical File System), используемая на Mac OS, и UFS2, используемая на FreeBSD. Эти файловые системы имеют свои характеристики и особенности, которые делают их подходящим выбором для определенных задач.

В целом, UFS-подобные файловые системы представляют собой эффективный и удобный способ организации и управления данными в Unix-подобных ОС. Выбор конкретной файловой системы зависит от требований и задачи, которую необходимо выполнить.

Кроме того, UFS-подобные файловые системы хорошо подходят для использования на серверах и в системах с высокой надежностью и безопасностью данных. Это связано с тем, что они предоставляют возможность создания резервных копий и восстановления данных, а также обеспечивают стабильную работу при сбоях и аварийных ситуациях.

Одним из главных недостатков UFS-подобных файловых систем является ограничение на размер файлового блока, который может быть использован для хранения данных. Наиболее распространенный размер блока - 4 Кб, что делает их неэффективными для работы с большими файлами или массивами данных.

В связи с этим были разработаны новые файловые системы, такие как XFS и JFS, которые предоставляют более гибкие возможности управления блоками данных и позволяют использовать большие блоки для хранения данных. Они также обеспечивают более высокую производительность и надежность работы с данными.

Таким образом, UFS-подобные файловые системы продолжают использоваться на многих Unix-подобных ОС, и они имеют свои преимущества и недостатки. Выбор конкретной файловой системы зависит от требований и задачи, которую необходимо выполнить, и необходимо учитывать все имеющиеся варианты для оптимального решения задачи.

1.3 • Анализ существующих решений в области файловых систем для Linux

Анализ существующих решений в области файловых систем для Linux является важным шагом при выборе наиболее подходящей файловой системы

для конкретной задачи. В этом пункте будут рассмотрены некоторые из самых популярных и распространенных файловых систем для Linux.

Ext4 (Fourth Extended Filesystem) - это файловая система, которая была разработана как улучшение предыдущей версии Ext3. Она предоставляет поддержку файлов и разделов большего размера, а также более эффективное использование пространства на диске. Ext4 также имеет множество функций, таких как journaling, delayed allocation и extents, которые обеспечивают большую безопасность и производительность работы с данными.

Btrfs (B-tree filesystem) - это новая файловая система, которая была разработана для использования в Linux. Btrfs предоставляет многочисленные функции, такие как снимки, копирование на запись, проверка целостности данных, резервное копирование и восстановление данных. Она также обеспечивает более высокую производительность, чем старые файловые системы, за счет использования технологий, таких как CoW (Copy-on-Write) и multithreading.

XFS (X File System) - это файловая система, которая была разработана для работы с большими объемами данных. XFS обеспечивает высокую производительность на больших файлах и массивах данных за счет использования многопоточной структуры и оптимизированного управления блоками данных. Она также обладает функциями, такими как journaling, delayed allocation и extents, которые обеспечивают безопасность и эффективность работы с данными.

ZFS (Zetabyte File System) - это файловая система, которая была разработана компанией Sun Microsystems и теперь поддерживается сообществом OpenZFS. ZFS предоставляет надежность и безопасность данных за счет использования технологий, таких как snapshotting, data checksumming и RAID-защита. Она также обеспечивает высокую производительность на больших объемах данных, так как оптимизирована для работы с массивами данных.

NTFS (New Technology File System) - это файловая система, которая была разработана компанией Microsoft и используется на операционных системах Windows. Она может быть использована в Linux с помощью spftware-решений, таких как ntfs-3g. NTFS обеспечивает поддержку файлов и дисков большого размера, а также функции безопасности и контроля доступа.

Кроме того, для специфических задач могут быть использованы другие файловые системы, такие как JFS (Journaling File System) для обеспечения надежности и безопасности работ, F2FS (Flash-Friendly File System) для работы с флеш-накопителями, или ReiserFS для оптимизации работы с маленькими файлами.

В целом, выбор конкретной файловой системы зависит от требований и задачи, которую необходимо выполнить. Необходимо учитывать все имеющиеся варианты и особенности каждой из файловых систем для выбора наиболее подходящего решения для конкретной задачи.

2.СТРУКТУРНОЕ ПРОЕКТРИРОВАНИЕ

Программный код был разбит на модули для более удобного использования и изменения функционала программы.

2.1 main.c

В этом файле написана функция main, которая парсит командную строку и, в зависимости от прописанных аргументов командной строки, вызывает определенные функции.

2.2 utils.c

Данный файл содержит в себе весь функционал по работе с файловой системой. Здесь прописаны функции change_mode, create, remove_file, is_path, listfile, listdir, move, recWalk, dirwalk,

2.3 Makefile

Makefile отвечает за сборку консольного приложения, перемещение исполняемого файла в директорию ~bin/ и очистку репозитория от объект-файлов.

3. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Для подробного рассмотрения кода и понимания программы ниже представлены алгоритмы по шагам основных. Также в приложении Б представлены блок-схемы функций.

3.1 Функция int main(int argc, char* argv[])

Шаг 1. Объявите переменную "c" типа char.

Шаг 2. Объявите переменную "option_index" типа int и инициализируйте ее значением 0.

Шаг 3. Создайте массив структур "option_types", содержащий опции, которые программа может принимать. Каждая структура содержит имя опции, требования к аргументам и символ, который будет использоваться для идентификации опции в командной строке.

Шаг 4. Начните цикл while, который будет выполняться до тех пор, пока функция getopt_long не вернет константу EOF (конец файла).

Шаг 5. В теле цикла используйте оператор switch, чтобы выполнить различные действия в зависимости от значения переменной "c".

Шаг 6. Если значение переменной "c" равно 'c', проверьте, что переданы оба аргумента, и если это не так, выведите сообщение об ошибке и верните значение 1. Если оба аргумента присутствуют, вызовите функцию "create" с указанными именами файла и директории.

Шаг 7. Если значение переменной "c" равно 'r', вызовите функцию "remove_file" с аргументом optarg.

Шаг 8. Если значение переменной "c" равно 'd', вызовите функцию "dirwalk" с аргументом optarg.

Шаг 9. Если значение переменной "c" равно 'l', вызовите функцию "listdir" с аргументом optarg.

Шаг 10. Если значение переменной "c" равно 'm', проверьте, что переданы оба аргумента, и если это не так, выведите сообщение об ошибке и

верните значение 1. Если оба аргумента присутствуют, вызовите функцию "move" с указанными именами файла и директории.

Шаг 11. Если значение переменной "c" равно 'h', проверьте, что переданы оба аргумента, и если это не так, выведите сообщение об ошибке и верните значение EXIT_FAILURE. Если оба аргумента присутствуют, вызовите функцию "change_mode" с указанным именем файла и режимом.

Шаг 12. Если значение переменной "c" не соответствует ни одной из опций, выведите сообщение об ошибке и завершите программу с помощью функции exit(1).

Шаг 13. Конец цикла while.

Шаг 14. Конец функции main.

3.2 Функция для перемещения файла , находящегося по пути filepath в директорию по пути destination int move(char* filepath, char* destination)

Шаг 1. . Передаваемый путь к файлу и назначение проверяются на соответствие корректному формату.

Шаг 2. Если переданный путь не является действительным путем к файлу, функция вызывает функцию perror() и завершает свою работу с кодом выхода 2.

Шаг 3. То же самое происходит, если переданный путь назначения также не является действительным путем.

Шаг 4. Создается массив символов dest размером 256 байт для хранения полного пути назначения файла.

Шаг 5. Функция snprintf() используется для формирования полного пути файла назначения путем объединения переданного пути назначения и пути файла.

Шаг 6. Функция rename() используется для перемещения файла из исходного пути в новый путь, используя созданный ранее полный путь к файлу.

Шаг 7. Если функция rename() вернула отрицательное значение, то значит возникла ошибка при перемещении файла. В этом случае вызывается функция perror() для вывода сообщения об ошибке, и функция завершает свою работу с кодом выхода 1.

Шаг 8. Если перемещение файла было успешно выполнено, то выводится сообщение о том, что файл успешно перемещен в указанную директорию.

Шаг 9. Функция завершает свою работу с кодом выхода 0, указывая на успешное завершение операции перемещения файла.

3.3 Функция int change_mode(char* filepath, char* mode_str) , меняющая модификаторы доступа файла по пути filepath на модификатор доступа mode_str

Шаг 1. Проверяем, является ли переданный путь корректным с помощью функции is_path.

Шаг 2. Если путь некорректный, выводим сообщение об ошибке через perror и завершаем программу с кодом 2.

Шаг 3. Используем функцию strtol для конвертации строки с режимом доступа в целочисленное значение типа __mode_t.

Шаг 4. Вызываем функцию chmod, передавая ей путь к файлу и новый режим доступа в качестве аргументов.

Шаг 5. Если chmod вернула -1 (ошибка), выводим сообщение об ошибке через perror и возвращаем код завершения EXIT_FAILURE.

Шаг 6. Если изменение режима доступа произошло успешно, выводим сообщение о его изменении через printf.

Шаг 7. Возвращаем код завершения EXIT_SUCCESS.

3.4 Функция int dirwalk(char* directory), рекурсивно выводящая все файлы в директории directory и ниже

Шаг 1. Объявление функции dirwalk() с аргументом directory типа char*.

Шаг 2. Проверка, является ли переданный параметр путём к директории с помощью функции is_path().

Шаг 3. Если переданный параметр не является путём к директории, то выводится сообщение об ошибке с использованием perror() и происходит завершение программы с кодом 2.

Шаг 4. Если переданный параметр является путём к директории, то происходит вызов функции recWalk(), которая выполнит рекурсивный обход директории и всех её поддиректорий.

Шаг 5. Функция dirwalk() завершается без возвращения значения.

3.5 Функция int is_path(char* path), проверяющая является ли path путем к файлу

Шаг 1. Функция is_path(char* path) принимает на вход строку path, которая содержит путь к файлу или каталогу.

Шаг 2. Создание переменной типа struct stat с именем st, которая будет использоваться для хранения информации о файле или каталоге.

Шаг 3. Вызов функции stat(path, &st), которая получает информацию о файле или каталоге по указанному пути и сохраняет эту информацию в структуре st.

Шаг 4. Проверка результата выполнения функции stat(path, &st). Если результат равен 0 (успешное выполнение), то возвратить 1 (путь существует).

Шаг 5. Если результат выполнения функции stat(path, &st) не равен 0, то возвратить 0 (путь не существует).

4. РУКОВОДСТОВО ПОЛЬЗОВАТЕЛЯ

4.1 Подготовка системы для работы с файловой системой ufs и сборка приложения

Для решение задачи я создал приложение ufs_si, которое может выполнять базовые функции системного интерфейса. Перед тем как его использовать, требуется подготовить вашу операционную систему.

По умолчанию, linux поддерживает файловую систему ufs только для чтения, однако если вы хотите выполнять операции записи данных, то вам придется изменить файл конфигурации и пересобрать ядро вашей системых[1]. Для этого вам понадобится скачать последнюю версию ядра[2], подтвердить целостность ядра, скопировать старый конфигурационный файл [3] и заменить в нем строку "# CONFIG_UFS_FS_WRITE is not set" на "CONFIG_UFS_FS_WRITE=y"[4]. После этого надо скомпилировать ядро и его модули, после чего установить новое ядро и внести изменения в grub.

После этих операций вы сможете читать и записывать файлы для файловой системы ufs.

Все эти шаги необходимо проделать для того, чтобы приложение корректно могло создавать, удалять и изменять файлы. Если вам нужны только операции чтения, то изменять ядро вашей операционной системы не обязательно.

Также требуется скомпилировать приложение ufs_si. Для этого зайдите в директорию с кодом программы, там должен находиться файл Makefile. Напишите команду make в терминал и приложение скомпилируется и будет готово к использованию. При компиляции могут потребоваться права суперпользователя, они нужны чтобы исполняемый файл перенести в директорию ~bin/ для того, чтобы вы могли вызывать приложение из разных директорий не прописывая путь к исполняемому файлу, а написав только название приложения.

4.2 Функционал приложения ufs_si и нюансы работы

Приложение ufs_si может выполнять следующие функции :

- Удалять файлы
- Создавать файлы
- Переносить файлы из одной директории в другую в пределах одной файловой системы
- Вывод всех файлов в директории с модификаторами доступа и некоторыми метаданными
- Рекурсивный вывод всех файлов в директории и ниже
- Изменение модификатора доступа файла

Каждой из этих функций соответствует собственный короткий и длинный параметры. Функции удаления, создания, переноса и изменения модификатора доступа файла требуют прав суперпользователя из-за записи данных.

4.2.1 Удаление файлов

Удаление файлов выполняется через короткий аргумент -r или длинный аргумент --remove. Формат использования :

sudo ufs_si -r <filepath>

sudo ufs_si --remote <filepath>

Команда удаляет файл по пути filepath и выводит сообщение об успешном выполнении задачи либо об ошибке. Удаление файлов должно вызываться с правами суперпользователя, иначе команда не будет выполнена.

4.2.2 Создание файлов

Создание файлов выполняется через короткий аргумент -c или длинный аргумент --create. Формат использования :

sudo ufs_si -c <filename> <directory>

sudo ufs_si --create <filename> <directory>

Команда создает файл с именем filename в директории directory и выводит сообщение об успешном выполнении задачи либо об ошибке. Если

требуется создать файл с определенным расширением, то в названии файла filename требуется прописать его расширение. В ином случае создается текстовый файл без расширения. Создание файлов должно вызываться с правами суперпользователя, иначе команда не будет выполнена.

4.2.3 Перенос файлов

Перенос файлов выполняется через короткий аргумент -m или длинный аргумент --move. Формат использования :

sudo ufs_si -m <filepath> <directory>

sudo ufs_si –move <filepath> <directory>

Команда переносит файл по пути filepath в директорию directory и выводит сообщение об успешном выполнении задачи либо об ошибке. Перенос файлов должен вызываться с правами суперпользователя, иначе команда не будет выполнена.

4.2.4 Вывод всех файлов в директории с модификаторами доступа и некоторыми метаданными

Вывод всех файлов в директории с модификаторами доступа и некоторыми метаданными выполняется через короткий аргумент -l или длинный аргумент --listdir. Формат использования :

ufs_si -l <directory>

ufs_si –listdir <directory>

Команда выводит все файлы в директории directory вместе с их модификаторами доступа, типами файлов, последним временем записи в файл и именами.

4.2.5 Рекурсивный вывод всех файлов в директории и ниже

Рекурсивный вывод всех файлов в директории и ниже по фаловому дереву выполняется через короткий аргумент -d или длинный аргумент --dirwalk. Формат использования :

ufs_si -d <directory>

ufs_si –dirwalk <directory>

Команда выводит все файлы в директории directory. Если при выводе программа встречает директорию, то она приостанавливает свою работу и начинает выводить файлы из этой директории. Таким образом команда выводит все файлы директории и все файлы ниже этой директории по файловому дереву.

4.2.6 Изменение модификатора доступа файла

Изменение файлов выполняется через короткий аргумент -h или длинный аргумент --chmod. Формат использования :

sudo ufs_si -h <filepath> <mode>

sudo ufs_si --chmod<filepath> <mode>

Команда меняет модификаторы доступа файла по пути filepath на модификаторы mode. Mode представляет собой 3 цифры, показывающие модификаторы доступа для автора файла, группы пользователей автора и остальным пользователям.

Три варианта записи прав пользователя[1]

двоичная	**восьмеричная**	**символьная**	**права на файл**	**права на каталог**
000	0	`---`	нет	нет
001	1	`--x`	выполнение	чтение свойств файлов
010	2	`-w-`	запись	нет
011	3	`-wx`	запись и выполнение	всё, кроме получения имени файлов
100	4	`r--`	чтение	чтение имён файлов
101	5	`r-x`	чтение и выполнение	доступ на чтение файлов/их свойств
110	6	`rw-`	чтение и запись	чтение имён файлов

111	7	rwx	все права	все права

Изменение модификатора доступа файла должно выполняться с правами суперпользователя, иначе команда не будет выполнена.

ЗАКЛЮЧЕНИЕ

В данном курсовом проекте была реализована программа ufs_si на основе выученного материала и знаний по дисциплине «Операционные системы и системное программирование ».

В заключение можно сказать, что системный интерфейс UFS-подобных файловых систем является важной составляющей функционирования операционной системы Linux. Использование данного интерфейса позволяет обеспечить более эффективную работу с файловой системой, а также повысить ее надежность и безопасность.

В ходе выполнения курсовой работы был проведен анализ основных принципов функционирования UFS-подобных файловых систем, изучены способы их монтирования и настройки в операционной системе Linux. Было произведено тестирование работы UFS-подобной файловой системы на реальном оборудовании, что подтвердило ее стабильность и возможности.

Таким образом, использование системного интерфейса UFS-подобных файловых систем в операционной системе Linux является актуальной задачей в современных условиях. Он позволяет повысить производительность, надежность и безопасность файловой системы, что является важным фактором для обеспечения стабильной работы операционной системы в целом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] cateee.net/ - информационный сервис – [Электронный ресурс]. – режим доступа : https://cateee.net/lkddb/web-lkddb/UFS_FS.html - [дата доступа] – 01.04.2023

[2] cyberciti.biz/ - информационный сервис – [Электронный ресурс]. – режим доступа : https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html - [дата доступа] – 12.04.2023

[3] www.youtube.com/ - информационный сервис – [Электронный ресурс]. – режим доступа : https://www.youtube.com/watch?v=1fdDNGU5s8I - [дата доступа] – 19.04.2023

[4] www.psx-place.com/ - информационный сервис – [Электронный ресурс]. – режим доступа : https://www.psx-place.com/threads/tutorial-hdd-mounting-and-decryption-on-linux.23308/ - [дата доступа] – 20.04.2023

[5] younglinux.info/ - информационный сервис – [Электронный ресурс]. – режим доступа : https://younglinux.info/bash/chmod - [дата доступа] – 01.05.2023

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода

\\ файл “main.c”

```
#define _DEFAULT_SOURCE

#include <getopt.h>
#include "utils.c"

int main(int argc, char* argv[])
{
    char c;
    int option_index = 0;
    struct option option_types[] = {
            {"create", required_argument, 0, 'c'},
            {"remove", required_argument, 0, 'r'},
            {"dirwalk", required_argument, 0, 'd'},
            {"listdir", required_argument, 0, 'l'},
            {"move", required_argument, 0, 'm'},
            {"chmod", required_argument, 0, 'h'}
    };

    while((c = getopt_long(argc, argv, "cr:d:l:mh",
                           option_types, &option_index)) != EOF)
    {
        switch (c)
        {
            case 'c':
                if (argc < 3) {
                   printf("Usage: %s <filename> <directory>\n", argv[0]);
                   return 1;
                }
                create(argv[argc - 2], argv[argc - 1]);
                break;
            case 'r':
                remove_file(optarg);
                break;
```

```
                case 'd':
                    dirwalk(optarg);
                    break;
                case 'l':
                    listdir(optarg);
                    break;
                case 'm':
                    if (argc < 3) { // Check if both arguments are provided
                        printf("Usage: sudo %s <file> <directory>\n", argv[1]);
                        return 1;
                    }
                    move(argv[argc - 2], argv[argc - 1]);
                    break;
                case 'h':
                    if (argc < 3) {
                         fprintf(stderr, "Usage: sudo %s <filepath> <mode>\n",
argv[1]);
                        return EXIT_FAILURE;
                    }
                    change_mode(argv[argc - 2], argv[argc - 1]);
                    break;
                default:
                    perror("Incorrect option");
                    exit(1);
                    break;
            }
        }
    }
```

\\ файл “utils.c”

```
#define _DEFAULT_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <dirent.h>
#include <string.h>
#include <getopt.h>
```

```c
#include <stdbool.h>
#include <unistd.h>
#include <time.h>

/* change_mode function sets file's permission mode given in argv[1] to mode given argv[2] */
int change_mode(char* filepath, char* mode_str) {

    if (!is_path( filepath))
    {
        perror(filepath);
        exit(2);
    }

    __mode_t mode = strtol(mode_str, NULL, 8);

    if (chmod(filepath, mode) == -1) {
        perror("chmod");
        return EXIT_FAILURE;
    }

    printf("Changed mode of file '%s' to %04o\n", filepath, mode);
    return EXIT_SUCCESS;
}

/* create function creates file with name given in argv[1] including file extension */
int create(char* filename, char* directory) {

    if (!is_path(directory))
    {
        perror(directory);
        exit(2);
    }

    char filepath[strlen(directory) + strlen(filename) + 2];
    sprintf(filepath, "%s/%s", directory, filename);
```

```
    FILE* fp = fopen(filepath, "w");
    if(fp == NULL) {
        printf("Error creating %s.\n", filename);
        return 1;
    }

    printf("%s created successfully in %s.\n", filename, directory);

    fclose(fp);
    return 0;
}

/* remove_file function deletes file given in argv[1] */
int remove_file(char* filepath) {
    if (!is_path( filepath))
    {
        perror(filepath);
        exit(2);
    }

    if(remove(filepath) == 0) {
        printf("%s deleted successfully.\n", filepath);
    } else {
        printf("Error deleting %s.\n", filepath);
    }
    return 0;
}

/* function is_path checks whether given char* is correct path or not. Return
1 if it is, 0 if it isn't , otherwise it's error*/
int is_path(char* path) {
    struct stat st;
    if (stat(path, &st) == 0) return 1;
    return 0;
}

/* listfile lists given file like ls -la */
void listfile(char* name)
```

```
{
    struct stat sb;
    char *modtime;


    char *filetype[] = {"?", "p", "c", "?", "d", "?", "b", "?", ".", "?",
"l", "?", "s"};

    if(stat(name, &sb) < 0)
    {
        perror(name);
        exit(2);
    }


    printf("%s", filetype[(sb.st_mode >> 12) & 017]);

    printf("%c%c%c%c%c%c%c%c%c",
            (sb.st_mode & S_IRUSR) ? 'r' : '-',
            (sb.st_mode & S_IWUSR) ? 'w' : '-',
            (sb.st_mode & S_IXUSR) ? 'x' : '-',
            (sb.st_mode & S_IRGRP) ? 'r' : '-',
            (sb.st_mode & S_IWGRP) ? 'w' : '-',
            (sb.st_mode & S_IXGRP) ? 'x' : '-',
            (sb.st_mode & S_IROTH) ? 'r' : '-',
            (sb.st_mode & S_IWOTH) ? 'w' : '-',
            (sb.st_mode & S_IXOTH) ? 'x' : '-');

    printf("%8ld", sb.st_size);

    modtime = ctime(&sb.st_mtime);
    modtime[strlen(modtime) - 1] = '\0';
    printf(" %s ", modtime);
    printf("%s\n", name);

}

/* lists all files in given directory like ls -la */
```

```
void listdir(char * directory)
{
    DIR *d;
    struct dirent * info;
    // if(argc != 2){
    //     fprintf(stderr, "usage: listdir dirname\n");
    //     exit(1);
    // }

    if (!is_path( directory))
    {
        perror(directory);
        exit(2);
    }

    chdir(directory);
    d = opendir(directory);

    while((info = readdir(d)) != NULL)
    {
        if (info->d_type == DT_LNK)
        {
            continue;
        }

        listfile(info->d_name);
    }

    closedir(d);
}

/* move functions moves file given in argv[1] to dest in argv[2] */
int move(char* filepath, char* destination) {

    if (!is_path( filepath))
    {
        perror(filepath);
        exit(2);
```

```
        }

        if (!is_path( destination))
        {
            perror(destination);
            exit(2);
        }

        // Construct the full path of the destination file
        char dest[256];
        snprintf(dest, sizeof(dest), "%s/%s", destination, filepath);

        // Move the file to the destination directory
        if (rename(filepath, dest) < 0) {
            perror("Error moving file");
            return 1;
        }

        printf("%s successfully moved to %s\n",filepath, destination);
        return 0;
    }

    void recWalk(const char* dir_name)
    {
        DIR *directory;
        /* namelist изначально NULL, тк scandir сам выделяет память, */
        struct dirent **namelist = NULL;
        int numOfFiles = 0;

        if ((directory = opendir(dir_name)) == NULL)
        {
            perror(dir_name);
            printf("opendir error\n");

            exit(1);
        }
        /* смена рабочей директории на ту, которую проходим, чтобы в дальнейшем
можно было
```

```
    открывать директорию с помощью только ее имени*/
    chdir(dir_name);


    numOfFiles = scandir(".", &namelist, NULL, alphasort);
    if(numOfFiles == -1)
    {
        perror("Scandir error");
        exit(1);
    }

    while (numOfFiles--)
    {
        printf("%s\n", namelist[numOfFiles]->d_name);

        if (namelist[numOfFiles]->d_type == DT_DIR
        && namelist[numOfFiles]->d_name[0] != '.')
        {
            recWalk((const char*)namelist[numOfFiles]->d_name);
        }
    }

    free(namelist);

    closedir(directory);
    chdir("..");
}

/* dirwalk function recursively traverses files in given directory*/
int dirwalk(char* directory)
{
    if (!is_path( directory))
    {
        perror(directory);
        exit(2);
    }

    recWalk(directory);
```

```
}
```

\\ файл “Makefile”

```
CC = gcc

CFLAGS = -c -std=c11 -pedantic -W -Wall -Wextra

.PHONY:all
all: start

start:

    sudo $(CC) main.c utils.c  $(CFLAGS)
    $(CC) main.o -o ~bin/ufs_si

clean:
    rm -rf main.o utils.o
```

ПРИЛОЖЕНИЕ Б

(обязательное)

ПРИЛОЖЕНИЕ В

(обязательное)

Ведомость документов.