РЕФЕРАТ

Отчёт по курсовому проекту содержит: 59 страниц, 21 рисунок, 15 таблиц, 3 приложения, 4 источника.

Объект исследования – операционная система, файловая система, межпроцессное взаимодействие.

Цель – спроектировать гипотетическую операционную систему, разработать эмуляцию отдельных её модулей.

Результат – проект гипотетической операционной системы, программы эмуляции файловой системы, командного интерпретатора и межпроцессного взаимодействия.

ОПЕРАЦИОННАЯ СИСТЕМА, ФАЙЛОВАЯ СИСТЕМА, МЕЖПРОЦЕССНОЕ ВЗАИМОДЕЙСТВИЕ, СЕМАФОРЫ, РАЗДЕЛЯЕМАЯ ПАМЯТЬ, КОМАНДНЫЙ ИНТЕРПРЕТАТОР , ТАБЛИЦА РАЗМЕЩЕНИЯ ФАЙЛОВ

СОДЕРЖАНИЕ

[Введение 5](#_Toc93378580)

[1 Структура проектируемой файловой системы 6](#_Toc93378581)

[1.1 Общая организация файловой системы 6](#_Toc93378582)

[1.2 Виртуальные страницы 9](#_Toc93378583)

[1.3 Команды для работы с фс 10](#_Toc93378584)

[1.4 Системные вызовы для работы с фс 12](#_Toc93378585)

[1.5 Способы организации файлов 15](#_Toc93378586)

[1.6 Алгоритмы работы некоторых системных вызовов фс 15](#_Toc93378587)

[2 Процессы в ос 17](#_Toc93378588)

[2.1 Приоритеты процессов 17](#_Toc93378589)

[2.2 Межпроцессное взаимодействие 18](#_Toc93378590)

[2.3 Свопинг процессов 19](#_Toc93378591)

[3 Режимы работы проектируемой ос 20](#_Toc93378592)

[3.1 Мультипрограммный режим работы ос 20](#_Toc93378593)

[3.2 Многопользовательская защита 20](#_Toc93378594)

[3.3 Интерактивный режим работы ос 21](#_Toc93378595)

[3.4 Пакетный режим работы ос 21](#_Toc93378596)

[4 Структура операционной системы 22](#_Toc93378597)

[4.1 Общая структура проектируемой ос 22](#_Toc93378598)

[4.2 Структура ядра проектируемой ос. Основные функции и назначение файловой подсистемы, подсистемы управления памятью и процессами, подсистемы управления устройствами 23](#_Toc93378599)

[5 Разработка программ эмуляции ос 24](#_Toc93378600)

[5.1 Описание программных средств 24](#_Toc93378601)

[5.2 Разработка фс 24](#_Toc93378602)

[5.3 Разработка командного интерпретатора 26](#_Toc93378603)

[5.4 Эмуляция межпроцессного взаимодействия средствами семафоров и разделяемой памяти 27](#_Toc93378604)

[6 Тестирование программы. Анализ результатов 29](#_Toc93378605)

[Выводы 31](#_Toc93378606)

[Перечень ссылок 32](#_Toc93378607)

[Приложение А. Техническое задание 33](#_Toc93378608)

[Приложение Б. Экранные формы, отображающие результаты работы программ эмуляции 36](#_Toc93378609)

[Приложение В. Листинг программы 38](#_Toc93378610)

# ВВЕДЕНИЕ

Операционные системы окружают нас повсюду – это основное программное обеспечение персональных компьютеров, серверов, мобильных устройств, сетевых устройств и даже современных автомобилей, телевизоров и прочего. Перечислять можно очень долго, ведь они требуются практически в каждой компьютерной системе.

Любой компьютер представляет собой связанную совокупность: процессора, памяти и устройств ввода-вывода.

Главное назначение ОС – управление ресурсами, а главные ресурсы, которыми она управляет, – это аппаратура компьютера. ОС управляет вычислительным процессом и информационным обменом между процессором, памятью, внешними устройствами.

Поскольку все устройства компьютера работают одновременно, ОС обеспечивает разделение ресурсов, предотвращая тем самым опасность возникновения конфликтных ситуаций между компонентами вычислительной системы, способных привести к сбою в работе, потере или искажении информации.

Полученные знания и навыки при проектировании гипотетической ОС являются основой для принятия профессиональных решений при проектировании средств управления в различной операционной обстановке.

# СТРУКТУРА ПРОЕКТИРУЕМОЙ ФАЙЛОВОЙ СИСТЕМЫ

Файловая система является важной частью любого накопителя информации. Она позволяет организовывать файловое пространство и работать с ней операционной системе.

Файловая система – система, которая контролирует и определяют, как будут храниться и именоваться данные на носителе или накопителе информации. От неё зависит способ хранения информации, формат данных и то, как они будут считываться и записываться.

Для разрабатываемой в данном курсовом проекте файловой системы были использованы различные особенности файловых систем семейства FAT (16 и 32).

## Общая организация файловой системы

Логическая структура диска, которую формирует проектируемая ФС, представлена в таблице 1.1.

Таблица 1.1 – Логическая структура диска

|  |  |  |
| --- | --- | --- |
| Служебные области | Суперблок | FAT1 |
| FAT2 | Область данных пользователей | Область данных |

Суперблок используется для хранения информации о конкретной файловой системе. Он является начальной точкой файловой системы. Информация, хранимая в суперблоке, используется для организации доступа к остальным данным на диске. При монтировании файловой системы суперблок считывается в память и находится там до ее отключения.

Структура суперблока проектируемой ФС представлена в таблице 1.2.

Таблица 1.2 – Структура суперблока

|  |  |
| --- | --- |
| Поле | Размер |
| Тип файловой системы | 5 байт |
| Размер кластера | 2 байта |
| Размер FAT таблицы | 2 байта |
| Количество FAT таблиц | 1 байт |
| Размер области пользователей | 2 байта |
| Размер области данных | 4 байта |
| Всего | 16 байт |

В таблице размещения файлов (FAT) хранится информация о кластерах логического диска. Каждому кластеру соответствует элемент таблицы FAT, содержащий информацию о том, свободен данный кластер или занят данными файла.

Структура одной записи таблицы FAT представлена в таблице 1.3.

Таблица 1.3 – Структура записи таблицы FAT

|  |  |
| --- | --- |
| Поле | Размер |
| Флаг кластера (свободен или завершает файл), либо указатель на следующий кластер файла | 2 байта |

Описание главных характеристик каждого файла и каталога проектируемой ФС находится в каталожных записях файлов. Каталожные записи размещаются в области данных в кластерах родительского каталога файла.

Структура каталожной записи представлена в таблице 1.4.

Таблица 1.4 – Структура каталожной записи

|  |  |
| --- | --- |
| Поле | Размер |
| Имя файла | 11 байт |
| Атрибуты | 1 байт |
| Зарезервировано | 3 байта |
| Права доступа | 1 байт |
| Дата и время создания | 8 байт |
| Номер первого кластера | 2 байта |
| Размер файла | 4 байта |
| Id владельца файла | 2 байта |
| Всего | 32 байта |

Атрибуты файла позволяют указать особенности поведения системы при работе с файлом. Так при одном и том же действии, результат выполнения может отличатся для файлов с разными атрибутами.

Структура поля «Атрибуты файла» представлена в таблице 1.5.

Таблица 1.5 – Структура поля «Атрибуты файла»

|  |  |
| --- | --- |
| Бит | Описание |
| 0 | Каталог |
| 1 | Скрытый |
| 2 | Только чтение |
| 3 | Зарезервировано |
| 4 | Зарезервировано |
| 5 | Зарезервировано |
| 6 | Зарезервировано |
| 7 | Зарезервировано |

Права доступа файлов позволяют определить какие группы пользователей могут выполнять определенные действия над файлами. Таким образом, права доступа играют важную роль в многопользовательской защите.

Структура поля «Права доступа» представлена в таблице 1.6.

Таблица 1.6 – Структура поля «Права доступа»

|  |  |
| --- | --- |
| Бит | Описание |
| 0 | Чтение файла владельцем |
| 1 | Запись файла владельцем |
| 2 | Исполнение файла владельцем |
| 3 | Чтение файла другими пользователями |
| 4 | Запись файла другими пользователями |
| 5 | Исполнение файла другими пользователями |
| 6 | Зарезервировано |
| 7 | Зарезервировано |

## Виртуальные страницы

Любая программная система имеет логическую модель памяти. Самая простая из них – совпадающая с физической, когда все программы имеют прямой доступ ко всему адресному пространству. При таком подходе программы имеют доступ ко всему адресному пространству. Они могут не только мешать друг другу, но и способны привести к сбою работы всей системы.

Виртуальная память – это концепция, которая позволяет уйти от использования физических адресов, используя вместо них виртуальные адреса. При этом вся виртуальная память делится на участки памяти постоянного размера, называемые страницами.

## Команды для работы с ФС

Одной из основных подсистем любой операционной системы является файловая система. Для управления файловыми системами используются целая группы команд, позволяющих создавать и уничтожать их, переименовывать, копировать, редактировать и т.п.

Для работы с проектируемой файловой системой используются команды, представленные в таблице 1.7.

Таблица 1.7 – Команды для работы с ФС

|  |  |  |
| --- | --- | --- |
| Имя команды | Аргументы | Описание |
| format | - | Форматирует диск |
| help | - | Выводит информацию о командах системы |
| adduser | Имя, пароль | Добавляет нового пользователя в систему |
| deluser | Имя | Удаляет указанного пользователя из системы |
| login | Имя, пароль | Авторизация пользователя |
| logout | - | Выход из сессии |
| crdir | Имя(путь) каталога | Создает новый каталог с заданных названием |
| opendir | Имя(путь) каталога | Открывает каталог с указанным названием |
| crfile | Имя(путь) файла | Создает новый файл с заданным названием |

Продолжение таблицы 1.7

|  |  |  |
| --- | --- | --- |
| fileread | Имя(путь) файла | Открывает файл с указанным названием на чтение |
| filewrite | Режим записи, Имя(путь) файла | Открывает файл с названием на запись |
| rmfile | Имя(путь) файла | Удаляет файл с указанным названием |
| rmdir | Имя(путь) каталога | Удаляет каталог с указанным названием |
| dirlist | - / Имя(путь) каталога | Выводит список файлов находящихся в указанном каталоге |
| userlist | - | Выводит список всех пользователей |
| rename | Имя(путь) файла, Новое имя | Изменяет название каталога или файла на новое |
| chmod | Имя(путь) файла, права доступа | Задает новые права доступа для указанного каталога или файла |
| chattr | Имя(путь) файла, атрибуты | Задает новые атрибуты для указанного каталога или файла |
| move | Имя(путь) файла, Имя(путь) каталога | Перемещает файл или каталог в указанный каталог |

## Системные вызовы для работы с ФС

Системный вызов позволяет приложению обратиться к операционной системе с просьбой выполнить то или иное действие, оформленное как процедура (или набор процедур) кодового сегмента ОС. Для прикладного программиста операционная система выглядит как некая библиотека, предоставляющая некоторый набор полезных функций, с помощью которых можно упростить прикладную программу или выполнить действия, запрещенные в пользовательском режиме.

Системные вызовы, использующиеся для работы проектируемой файловой системы, представлены в таблице 1.8.

Таблица 1.8 – Системные вызовы для работы ФС

|  |  |  |  |
| --- | --- | --- | --- |
| Вызов | Аргументы | Результат | Описание |
| DiskFormatCommand | - | Task.Comp  letedTask | Форматирование диска |
| AddUserCommand | username, password | Task.Comp  letedTask | Добавление пользователя с именем username и паролем password |
| LogInCommand | username, password | Task.Comp  letedTask | Авторизация в системе под именем username |
| LogOutCommand | - | Task.Comp  letedTask | Выход пользователя из системы |
| CreateDirCommand | dirname | Task.Comp  letedTask | Создание нового каталога c именем dirname |

Продолжение таблицы 1.8

|  |  |  |  |
| --- | --- | --- | --- |
| OpenDirCommand | dirname | Task.Comp  letedTask | Назначает текущий  каталог каталогом с  именем dirname |
| DirListCommand | -/dirname | Список файлов | Выводит список всех файлов каталога dirname или текущего каталога |
| RenameCommand | dirname/ filename, new name | Task.Comp  letedTask | Изменяет имя каталога или файла на имя new name |
| ChangeAttributes  Command | dirname/ filename, attributes | Task.Comp  letedTask | Изменяет атрибуты каталога или файла на attributes |
| ChangeRightsCommand | dirname/ filename, rights | Task.Comp  letedTask | Изменяет права доступа каталога или файла на rights |
| UserListCommand | - | Список  Users | Выводит имена всех существующих пользователей |
| DeleteUserCommand | username | Task.Comp  letedTask | Удаляет пользователя с именем username из системы |

Продолжение таблицы 1.8

|  |  |  |  |
| --- | --- | --- | --- |
| CreateFileCommand | filename | Task.Comp  letedTask | Создает новый файл с именем filename |
| FileWriteCommand | filename | Task.Comp  letedTask | Открывает файл с именем filename в режиме записи |
| FileReadCommand | filename | string | Открывает файл с именем filename в режиме чтения |
| RemoveFileCommand | filename | Task.Comp  letedTask | Удаляет файл с именем filename |
| RemoveDirCommand | dirname | Task.Comp  letedTask | Удаляет каталог с именем dirname и все файлы внутри него |
| MoveCommand | filename/ dirname, dirname | Task.Comp  letedTask | Перемещает файл или каталог в каталог с именем dirname |

## Способы организации файлов

Так как прототипом проектируемой файловой системы была выбрана файловая система семейства FAT, то для организации файлов и доступа к ним используются так называемые FAT-таблицы (таблицы размещения файлов).

Для организации доступа к файлу ФС должна иметь сведения о номерах кластеров, где размещается каждый файл или данные этого файла. Номер каждого кластера соответствует номеру ячейки FAT-таблицы.

Значение ячейки таблицы равное 0 означает, что кластер свободен и в него можно записывать данные, значение FFFF – кластер занят и является концом одного из файлов. В других случаях последовательность байт показывает номер следующего кластера файла.

Проектируемая файловая система является многоуровневой.

## Алгоритмы работы некоторых системных вызовов ФС

Алгоритм работы команды crfile (создание файла), изображён на рисунке 1.1.

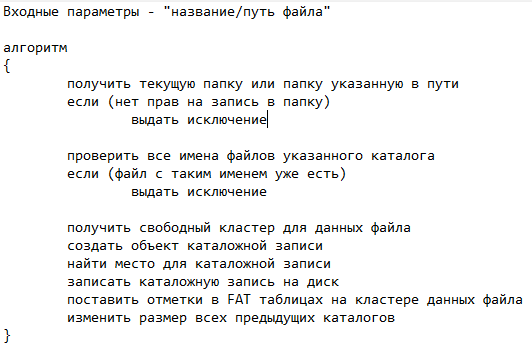


Рисунок 1.1 – Алгоритм работы команды crfile

Алгоритм работы команды rename (изменение имени), изображён на рисунке 1.2.

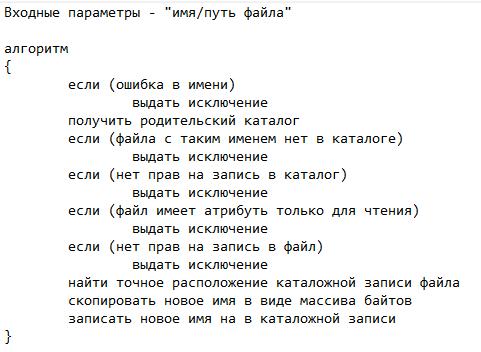


Рисунок 1.2 – Алгоритм работы команды rename

Алгоритм работы команды adduser (добавление пользователя), изображён на рисунке 1.3.

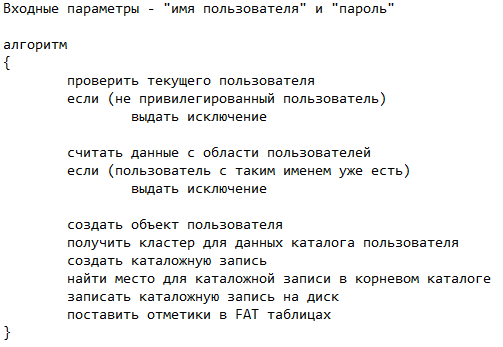


Рисунок 1.3 – Алгоритм работы команды adduser

# ПРОЦЕССЫ В ОС

Процесс (задача) – программа в стадии выполнения. Программа – это статический объект, представляющий собой файл с кодами и данными. Процесс – это динамический объект, который возникает в операционной системе после того, как пользователь или сама операционная система решает «запустить программу на выполнение», то есть создать новую единицу вычислительной работы.

## Приоритеты процессов

Приоритет процесса определяет, как часто данный процесс, по сравнению с другими процессами, стоящими в очереди на выполнение процессора, будет исполняться процессором, а если точнее от приоритета зависит квант времени процесса – время, в течение которого процесс может выполняться, прежде чем планировщик его прервет.

Относительные приоритеты – Относительный приоритет не изменяется самой системой системой на протяжении жизни процесса (но может изменятся пользователями). В системах, в которых планирование осуществляется на основе относительных приоритетов, минимизируются затраты на переключения процессора с одной работы на другую. С другой стороны, здесь могут возникать ситуации, когда одна задача занимает процессор долгое время.

Статические приоритеты – не изменяющиеся со временем приоритеты. Они присваиваются процессам до начала их выполнения и остаются постоянными. Механизмы статической приоритетности легко реализовать. Однако они не реагируют на изменения в окружающей ситуации, изменения, которые могут сделать желательной корректировку приоритетов.

## Межпроцессное взаимодействие

Межпроцессное взаимодействие – это обмен данными между потоками одного или разных процессов. Реализуется посредством механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС и реализующим новые возможности IPC. Может осуществляться как на одном компьютере, так и между несколькими компьютерами сети.

Для проектируемой системы были использованы следующие средства межпроцессного взаимодействия: семафоры и разделяемая память.

Семафор – примитив синхронизации работы процессов и потоков, в основе которого лежит счётчик, над которым можно производить две атомарные операции: увеличение и уменьшение значения на единицу, при этом операция уменьшения для нулевого значения счётчика является блокирующейся. Служит для построения более сложных механизмов синхронизации и используется для синхронизации параллельно работающих задач, для защиты передачи данных через разделяемую память, для защиты критических секций, а также для управления доступом к аппаратному обеспечению.

Разделяемая (общая) память – Один из самых простых методов взаимодействия между процессами. Разделяемую память применяют для того, чтобы увеличить скорость прохождения данных между процессами. В обычной ситуации обмен информацией между процессами проходит через ядро. Техника разделяемой памяти позволяет осуществить обмен информацией не через ядро, а используя некоторую часть виртуального адресного пространства, куда помещаются и откуда считываются данные.

Общая память позволяет двум или более процессам обращаться к одной и той же области памяти. Когда один процесс изменяет память, все другие процессы могут считать изменения из памяти.

Алгоритм работы разделяемой памяти представлен на рисунке 2.1.

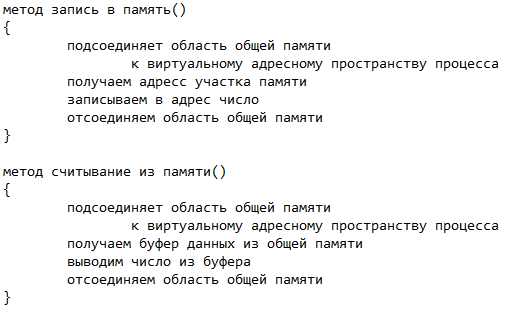


Рисунок 2.1 – Алгоритм работы разделяемой памяти

## Свопинг процессов

Свопинг процессов - в это процесс выгрузки редко используемых областей виртуального адресного пространства программы и/или всей программы на диск или другое устройство внешней памяти. Оперативная память, занимаемая процессом, освобождается.

Свопинг процессов может использоваться как средство реализации абсолютных приоритетов при отсутствии виртуальной памяти либо инициироваться операцией на внешнем устройстве, требующей значительных временных затрат.

# РЕЖИМЫ РАБОТЫ ПРОЕКТИРУЕМОЙ ОС

## Мультипрограммный режим работы ОС

Мультипрограммирование, или многозадачность, – это способ организации вычислительного процесса, при котором на одном процессоре попеременно выполняются сразу несколько программ. Эти программы совместно используют не только процессор, но и другие ресурсы компьютера: оперативную и внешнюю память, устройства ввода-вывода, данные. Мультипрограммирование призвано повысить эффективность использования вычислительной системы, однако эффективность может пониматься по-разному. Наиболее характерными критериями эффективности вычислительных систем являются: пропускная способность; удобство работы пользователей; реактивность системы.

## Многопользовательская защита

Многопользовательская защита – средства ОС, обеспечивающие распознавание разных пользователей и различные уровни прав, а также установку и защиту прав собственности на личные данные.

Многопользовательская защита позволяет защищать разных пользователей вычислительной машины, работающей под управлением ОС, от взаимного влияния.

Чтобы начать работать, пользователь должен "войти" в систему, введя свое учетное имя и, возможно, пароль. Пользователь, зарегистрированный в учетных файлах системы и, следовательно, имеющий учетное имя, называется зарегистрированным пользователем системы.

Регистрацию новых пользователей выполняет администратор системы.

## Интерактивный режим работы ОС

Интерактивный режим предусматривает непосредственное взаимодействие пользователя с информационно-вычислительной системой, может носить характер запроса (как правило, регламентированного) или диалога с ЭВМ. В этом режиме необходимо обеспечить приемлемое для пользователя время исполнения команды (как правило, в пределах нескольких секунд).

## Пакетный режим работы ОС

Пакетным называется режим работы компьютера, реализующий многозадачность (мультипрограммность), когда система обрабатывает заранее сформированный пакет заданий без вмешательства пользователя в процесс обработки.

При этом система сама распределяет задачи и процессы, оптимизируя загрузки ресурсов вычислительной системы и время выполнения.

В пакетном режиме обычно решались наиболее сложные и трудоемкие задачи, которые значительно загружают систему, и их стремились вывести за рамки основного рабочего времени организации — например, на ночь или на выходные.

# СТРУКТУРА ОПЕРАЦИОННОЙ СИСТЕМЫ

## Общая структура проектируемой ОС

Общая структура операционной системы изображена на рисунке 4.1.

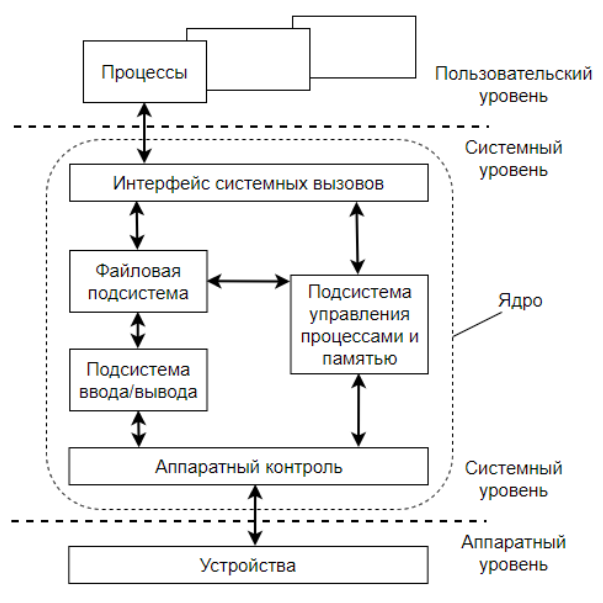


Рисунок 4.1 – Общая структура операционной системы

Ядро ОС – это управляющая часть ОС, постоянно находящаяся в оперативной памяти. Ядро ОС обеспечивает базовую функциональность ОС, создает процессы и управляет ими, распределяет память, обеспечивает доступ к файлам и периферийным устройствам. Взаимодействие прикладных задач с ядром происходит посредством стандартного интерфейса системных вызовов.

Интерфейс системных вызовов представляет собой набор услуг ядра и определяет формат запросов на услуги.

Процесс запрашивает услугу посредством системного вызов определенной процедуры ядра. Ядро от имени процесса выполняет запрос и возвращает процессу необходимые данные.

## Структура ядра проектируемой ОС. Основные функции и назначение файловой подсистемы, подсистемы управления памятью и процессами, подсистемы управления устройствами

Ядро операционной системы состоит трёх основных модулей:

– файловая подсистема;

– подсистема управления процессами и памятью;

– подсистема ввода-вывода.

Файловая подсистема обеспечивает интерфейс доступа к данным, расположенным на накопителяъ, и к периферийным устройствам. Она контролирует права доступа к файлу, выполняет операции размещения и удаления файла, а также выполняет запись или чтение данных файла. Запущенная на выполнение программа порождает в системе один или более процессов.

Подсистема управления процессами и памятью контролирует:

– создание и удаление процессов;

– распределение системных ресурсов между процессами;

– синхронизацию процессов;

– межпроцессное взаимодействие.

Подсистема ввода-вывода выполняет запросы файловой подсистемы и подсистемы управления процессами для доступа к периферийным устройствам. Она взаимодействует с драйверами устройств – специальными модулями ядра, непосредственно обслуживающими внешние устройства.

# РАЗРАБОТКА ПРОГРАММ ЭМУЛЯЦИИ ОС

## Описание программных средств

Для разработки программы эмуляции файловой системы был выбран язык программирования C#. Для разработки программы эмуляции межпроцессного взаимодействия был выбран язык C++. Разработка файловой системы велась c помощью интегрированной среды разработки Visual Studio 2022. Разработка эмулятора межпроцессного взаимодействия проводилась с помощью среды разработки Visual Studio Code.

## Разработка ФС

Класс файловой системы содержит основной функционал для работы с диском (файлом). Здесь реализованы такие методы, как: запись, чтение, поиск файлов и кластеров и т.д.

Диаграмма классов модуля файловой системы изображена на рисунке 5.1.

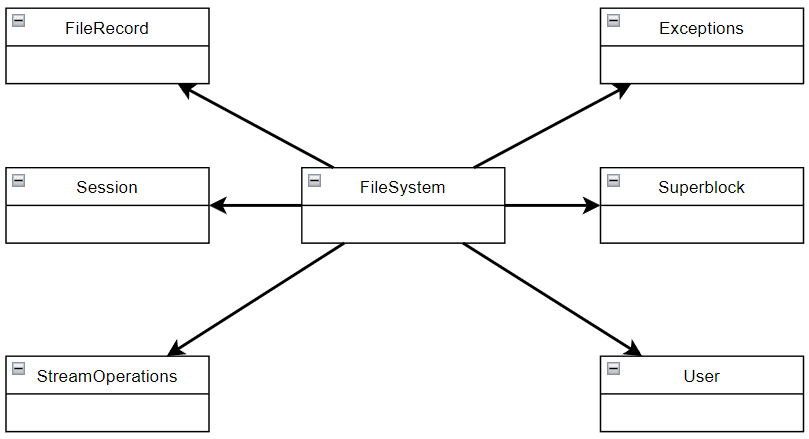


Рисунок 5.1 – Диаграмма классов модуля файловой системы

Методы класса FileSystem перечислены в таблице 5.1.

Таблица 5.1 – Методы класса FileSystem

|  |  |
| --- | --- |
| Имя метода | Описание |
| DiskFormat | Форматирование ФС |
| WriteData | Запись данных на диск |
| ReadData | Чтение данных с диска |
| FindLastCluster | Найти последний кластер файла или каталога |
| GetNextCluster | Получить следующий кластер файла или каталога |
| GetCluster | Получить свободный кластер в FAT таблице |
| GetAllFiles | Вывести список всех файлов указанного каталога |
| FindFile | Найти файл и расположение его данных в кластере |
| GetDir | Получить данные каталога |
| ResizeDirectories | Изменить размер каталогов |

Методы класса StreamOperations перечислены в таблице 5.2.

Таблица 5.2 – Методы класса StreamOperations

|  |  |
| --- | --- |
| Имя метода | Описание |
| Open | Открытие файла на чтение и запись |
| Write | Запись в файл |
| Read | Чтение файла |

Методы класса Session перечислены в таблице 5.3.

Таблица 5.3 – Методы класса Session

|  |  |
| --- | --- |
| Имя метода | Описание |
| RequestForAccess | Запрос прав администратора |
| isWritable | Проверка доступа к объекта для записи |
| isReadable | Проверка доступа к объекта для чтения |

## Разработка командного интерпретатора

Для общения с операционной системой пользователю предоставляется текстовая консоль.

Командный интерпретатор – это программа, которая в непрерывном цикле ожидает ввода команд. При вводе команды реализовывается ее синтаксический разбор (существование имени команды, наличие всех обязательных параметров и проверка параметров на их корректность).

Диаграмма классов модуля командного интерпретатора изображена на рисунке 5.2.

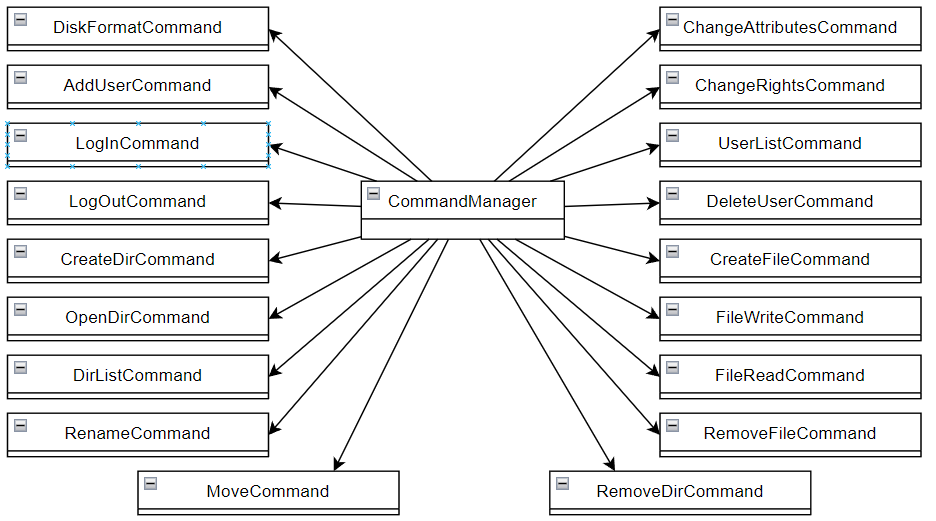


Рисунок 5.2 – Диаграмма классов модуля командного интерпретатора

Методы класса CommandManager перечислены в таблице 5.4.

Таблица 5.4 – Методы класса CommandManager

|  |  |
| --- | --- |
| Имя метода | Описание |
| Handle | Обработка введенной команды |

Класс CommandManager содержит список всех доступных команд операционной системы. При вводе данных пользователем, он выбирает нужную и приводит её в исполнение в методе Handle. Все команды для работы с файловой системой содержат в себе объект класса FileSystem. Каждая команда реализует интерфейс ICommand, который содержит методы, перечисленные в таблице 5.5.

Таблица 5.5 – Методы интерфейса ICommand

|  |  |
| --- | --- |
| Имя метода | Описание |
| isExecutable | Сверяет команду введенную пользователем с шаблоном существующей команды |
| Run | Выполняет команду |
| Info | Выводит информацию о команде |

## Эмуляция межпроцессного взаимодействия средствами семафоров и разделяемой памяти

Для демонстрации эмуляции межпроцессного взаимодействия была разработана программы, с помощью которой двое и более процессов обмениваются автоматически сгенерированными числами сообщениями через средства разделяемой памяти.

При этом для синхронизации входов процессов в критические области (запись и чтение из памяти) программа использует семафоры.

Диаграмма классов эмулятора межпроцессного взаимодействия изображена на рисунке 5.3.

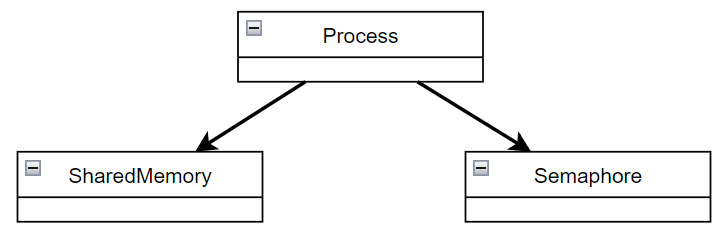


Рисунок 5.3 – Диаграмма классов эмулятора межпроцессного взаимодействия

Методы класса Process представлены в таблице 5.6

Таблица 5.6 – Методы класса Process

|  |  |
| --- | --- |
| Имя метода | Описание |
| CreateSharedArea | Создает участок общей памяти по заданному ключу |
| SetId | Установка Id созданного участка для подключения |
| StartAutoInteraction | Запуск взаимодействия процессов |

Методы класса SharedMemory представлены в таблице 5.7

Таблица 5.7 – Методы класса Process

|  |  |
| --- | --- |
| Имя метода | Описание |
| Attach | Подсоединяет область общей памяти к виртуальному адресному пространству процесса |
| SetId | Установка Id созданного участка для подключения |
| Detach | Отсоединят область общей памяти от виртуального адресного пространства процесса |
| Write | Записывает сообщение в память |
| Read | Читает сообщение из памяти |
| Free | Освобождает память от данных |

# ТЕСТИРОВАНИЕ ПРОГРАММЫ. АНАЛИЗ РЕЗУЛЬТАТОВ

Разработанные программные модули для эмуляции операционной системы корректно обрабатывают возможные исключительные ситуации, которые могут возникнуть при неверных действиях пользователя или при некорректном вводе данных.

Реакции системы на возникновения исключительных ситуаций изображены на рисунках 6.1-6.6.

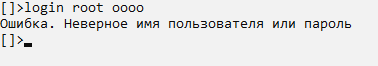


Рисунок 6.1 – Попытка авторизации с неверными данными



Рисунок 6.2 – Попытка добавления пользователя без прав администратора



Рисунок 6.3 – Открытие несуществующего файла на чтение

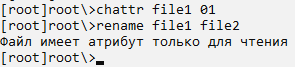


Рисунок 6.4 – Попытка изменения файла с атрибутом “Только для чтения”



Рисунок 6.5 – Попытка открытия файла на запись без соответствующих прав



Рисунок 6.6 – Неверный ввод команды

# ВЫВОДЫ

В большинстве вычислительных систем операционная система является основной, наиболее важной (а иногда и единственной) частью системного программного обеспечения. С 1990-х годов наиболее распространёнными операционными системами являются системы семейства Windows и системы класса UNIX (особенно Linux и Mac OS). С 2000-х большое распространение получили мобильные компьютеры (смартфоны и планшеты) и с ними ОС Android и iOS.

ОС является необходимой составляющей ПО ПК, без нее компьютер не может работать в принципе.

В ходе выполнения курсового проекта были изучены и закреплены на практике основные концепции работы операционной системы.

В результате выполнения курсового проекта была спроектирована гипотетическая операционная система, были разработаны отдельные её модули, а именно: файловая система, командный интерпретатор и эмулятор межпроцессного взаимодействия средствами разделяемой памяти и семафоров.

Разработанная система может быть использована для демонстрации возможностей работы операционной системы.

# ПЕРЕЧЕНЬ ССЫЛОК

1. FAT // Википедия. [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/FAT

2. Гордеев А. В. Операционные системы: Учебник для вузов. 2-е изд. – СПб.: Питер, 2007. – 416 с.: ил.

3. Файловая система FAT // Revol. [Электронный ресурс]. – Режим доступа: http://revol.ponocom.ru/-d/ReVoL-FAT-2003.pdf

4. Межпроцессное взаимодействие // Википедия. [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Межпроцессное

\_взаимодействие

# ПРИЛОЖЕНИЕ А

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

# ПРИЛОЖЕНИЕ Б

ЭКРАННЫЕ ФОРМЫ, ОТОБРАЖАЮЩИЕ РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММ ЭМУЛЯЦИИ



Рисунок Б.1 – Создание файла

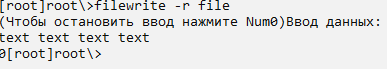


Рисунок Б.2 – Запись в файл



Рисунок Б.3 – Чтение файла

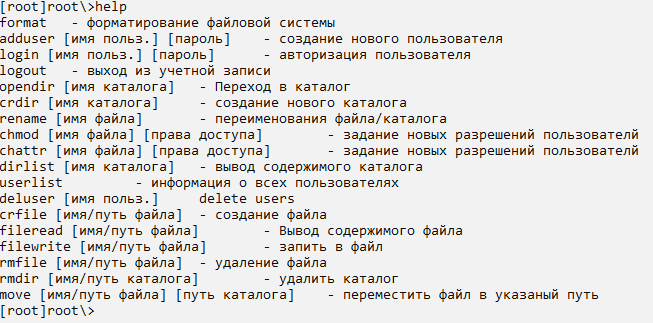


Рисунок Б.4 – Команда help

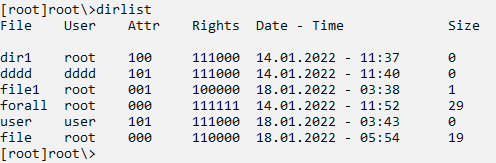


Рисунок Б.5 – Вывод списка файлов каталога

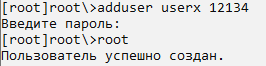


Рисунок Б.6 – Добавление пользователя

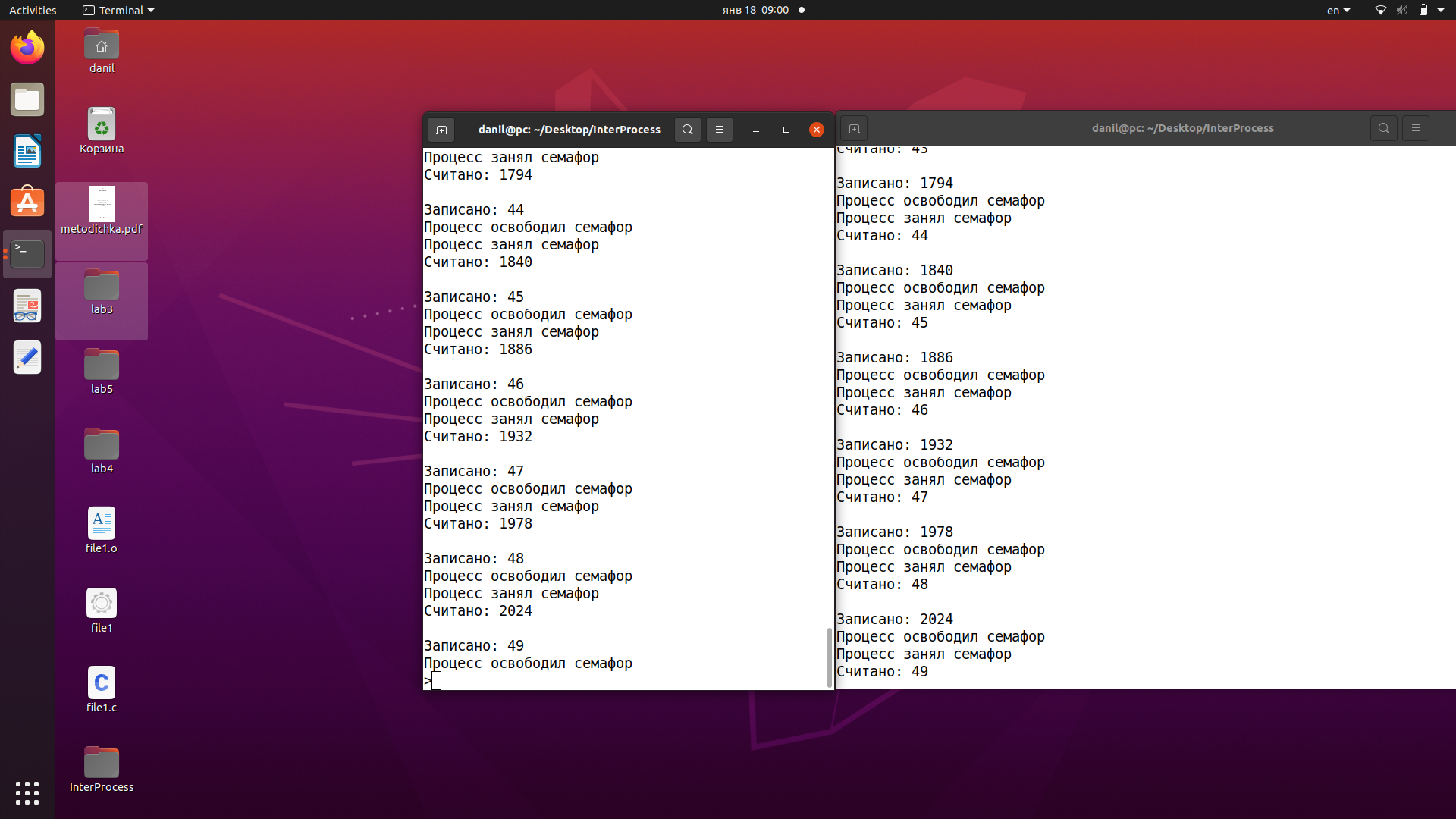


Рисунок Б.7 – Обмен данными между процессами с помощью общем памяти

# ПРИЛОЖЕНИЕ В

ЛИСТИНГ ПРОГРАММ

Program.cs

using Chaos.SystemComponents;

using Chaos.SystemComponents.Commands;

using Chaos.Tools;

var session = new Session();

var filesys = new FileSystem(new StreamOperations(), session);

List<ICommand> commands = new List<ICommand>()

{

new DiskFormatCommand(filesys,session),

new AddUserCommand(filesys,session),

new LogInCommand(filesys,session),

new LogOutCommand(filesys,session),

new OpenDirCommand(filesys,session),

new CreateDirCommand(filesys,session),

new RenameCommand(filesys,session),

new ChangeRightsCommand(filesys,session),

new ChangeAttributesCommand(filesys,session),

new DirListCommand(filesys,session),

new UserListCommand(filesys),

new DeleteUserCommand(filesys,session),

new CreateFileCommand(filesys,session),

new FileReadCommand(filesys,session),

new FileWriteCommand(filesys,session),

new RemoveFileCommand(filesys,session),

new RemoveDirCommand(filesys,session),

new MoveCommand(filesys,session)

};

var handler = new CommandManager(commands);

while (true)

{

Console.Write(session.ToString());

await handler.Handle(Console.ReadLine());

}

CommandManager.cs

using Chaos.SystemComponents.Commands;

namespace Chaos.SystemComponents

{

internal class CommandManager

{

private readonly IEnumerable<ICommand> commandlist;

public CommandManager(IEnumerable<ICommand> cmdlist)

{

commandlist = cmdlist;

}

public async Task Handle(string? command)

{

if (command == null) return;

if (command == "help")

{

commandlist.ToList().ForEach(a => a.Info()); return;

}

var process = commandlist.FirstOrDefault(a => a.isExecutable(command));

if (process != null)

{

try

{

await process.Run(command);

}

catch (Exception ex)

{ Console.WriteLine(ex.Message);

}

}

else Console.WriteLine($"{command}: неизвестная команда");

}

}

}

FileSystem.cs

using System.Collections;

using Chaos.Structures;

using Chaos.Tools;

using Chaos.Tools.Exceptions;

namespace Chaos.SystemComponents

{

internal class FileSystem

{

private readonly string disk = "disk";

private readonly StreamOperations stream;

private readonly Session currentSession;

public int EOF { get; private set; } = 65535;

public FileSystem(StreamOperations stream, Session session)

{

currentSession = session;

this.stream = stream;

if (!File.Exists(disk))

{

stream.Open(disk);

DiskFormat().GetAwaiter();

}

else stream.Open(disk);

}

public async Task DiskFormat()

{

await stream.Write(new byte[Superblock.FSSize], 0);

var i = 0;

await stream.Write(Superblock.Title.ToBytes(), i);

i += 5;

await stream.Write(Superblock.ClusterSize.ToBytes(), i);

i += 2;

await stream.Write(Superblock.FATSize.ToBytes(), i);

i += 2;

await stream.Write(Superblock.FATCount.ToBytes(), i);

i += 1;

await stream.Write(Superblock.UsersAreaSize.ToBytes(), i);

i += 2;

await stream.Write(Superblock.DataAreaSize.ToBytes(), i);

var rootuser = new User() { Id = 1, Name = "root", Password = "root" };

await stream.Write((byte[])rootuser, Superblock.UsersAreaStart);

var rootDir = new FileRecord()

{

Name = rootuser.Name,

Attributes = 160,

Rights = 252,

DateTime = DateTime.Now.ToLong(),

Cluster = 1,

Size = 0,

UserId = rootuser.Id

};

await stream.Write(EOF.ToBytes(), Superblock.TablesStart);

await stream.Write(EOF.ToBytes(), Superblock.TablesStart + Superblock.FATSize);

await stream.Write(EOF.ToBytes(), Superblock.TablesStart + 2);

await stream.Write(EOF.ToBytes(), Superblock.TablesStart + 2 + Superblock.FATSize);

await stream.Write((byte[])rootDir, Superblock.DataAreaStart);

}

public async Task WriteData(byte[] array, int startPosition, int endPosition)

{

for (int i = startPosition; i <= endPosition; i += array.Length)

{

var arr = new byte[array.Length];

await stream.Read(arr, i);

bool isClear = true;

for (int j = 0; j < arr.Length; j++)

{

if (arr[j] != '\0')

{

isClear = false;

break;

}

}

if (isClear)

{

await stream.Write(array, i);

return;

}

}

throw new NoFreeSpaceException();

}

public async Task<short> FindLastCluster(short start)

{

var next = start;

var cell = new byte[2];

do

{

start = next;

await stream.Read(cell, Superblock.TablesStart + start \* 2);

next = cell.ToShort();

} while (next != -1);

return start;

}

public async Task<short> GetNextCluster(short start)

{

var cell = new byte[2];

await stream.Read(cell, Superblock.TablesStart + start \* 2);

var next = cell.ToShort();

if (next == -1)

return start;

else

return next;

}

public async Task WriteData(byte[] array, int position)

{

await stream.Write(array, position);

}

public async Task<short> GetCluster()

{

for (int i = Superblock.TablesStart + 4; i < Superblock.TablesStart + Superblock.FATSize; i += 2)

{

var data = new byte[2];

await stream.Read(data, i);

bool empty = true;

for (int j = 0; j < data.Length; j++)

{

if (data[j] != 0)

{

empty = false;

break;

}

}

if (empty)

return (short)((i - Superblock.TablesStart) / 2);

}

throw new NoFreeClusterException();

}

public async Task<byte[]> ReadData(int startPosition, int size)

{

var bytes = new byte[size];

await stream.Read(bytes, startPosition);

return bytes;

}

public async Task<List<FileRecord>> GetAllFiles(short cluster)

{

var allfiles = new List<FileRecord>();

short next;

while (true)

{

var data = await ReadData(Superblock.DataAreaStart + (cluster - 1) \*

Superblock.ClusterSize, Superblock.ClusterSize);

var list = data.AsFileList();

allfiles.AddRange(list);

next = await GetNextCluster(cluster);

if (next == cluster)

return allfiles;

else

cluster = next;

}

}

public async Task<ArrayList> FindFile(string name, short cluster)

{

ArrayList info = new ArrayList();

short next = cluster;

do

{

cluster = next;

var data = await ReadData(Superblock.DataAreaStart + (cluster - 1) \*

Superblock.ClusterSize, Superblock.ClusterSize);

var list = data.AsFileList();

var file = list.Find(x => x.Name == name);

if (file == null)

next = await GetNextCluster(cluster);

else

{

info.Add(file);

info.Add(cluster);

info.Add(list.IndexOf(file));

return info;

}

} while (cluster != next);

return null;

}

public async Task<FileRecord> GetDir(string filepath)

{

FileRecord? item = null;

var directoryArray = filepath.Split('\\');

var data = await ReadData(Superblock.DataAreaStart, Superblock.ClusterSize);

var list = await GetAllFiles(1);

if (filepath == "")

{

return new FileRecord() { Name = "" };

}

for (int i = 0; i < directoryArray.Length; i++)

{

item = list.Where(x => x.Name == directoryArray[i] &&

x.Attributes.AsAttributesArray()[0]).FirstOrDefault();

if (item == null) throw new NoFileException();

currentSession.isReadable(item);

var cluster = item?.Cluster ?? throw new NoFileException();

list = await GetAllFiles(cluster);

}

if (item == null) throw new NoFileException();

return item;

}

public async Task ResizeDirectories(string path,int changeSize)

{

if (path == null || path == String.Empty)

path = currentSession.CurrentPath.Remove(currentSession.CurrentPath.Length - 1);

var dirArr = path.Split('\\');

path =dirArr[0];

for (int i = 0; i < dirArr.Length-1; i++)

{

var dir = await GetDir(path);

if (i + 1 == dirArr.Length)

break;

var nextdir = await FindFile(dirArr[i+1],dir.Cluster);

if (nextdir != null)

{

((FileRecord)nextdir[0]).Size = ((FileRecord)nextdir[0]).Size + changeSize;

await WriteData((byte[])(FileRecord)nextdir[0], Superblock.DataAreaStart + ((short)nextdir[1] - 1) \*

Superblock.ClusterSize + (int)nextdir[2] \* FileRecord.RecordSize);

}

path += "\\"+dirArr[i + 1];

}

}

}

}

FileRecord.cs

using Chaos.Tools;

namespace Chaos.Structures

{

internal class FileRecord

{

public static int RecordSize => 32;

public string Name { get; set; }

public byte Attributes { get; set; }

public byte Rights { get; set; }

public long DateTime { get; set; }

public short Cluster { get; set; }

public int Size { get; set; }

public short UserId { get; set; }

public static explicit operator byte[](FileRecord file)

{

var arr = new byte[RecordSize];

file.Name.ToBytes().CopyTo(arr, 0);

arr[11] = file.Attributes;

arr[15] = file.Rights;

file.DateTime.ToBytes().CopyTo(arr, 16);

file.Cluster.ToBytes().CopyTo(arr, 24);

file.Size.ToBytes().CopyTo(arr, 26);

file.UserId.ToBytes().CopyTo(arr, 30);

return arr;

}

public string ToString(List<User> userList)

{

if (Cluster == 1 && Name == "root")

return "";

var str = string.Empty;

str += Name + "\t";

str += (userList.FirstOrDefault(x => x.Id ==UserId)?.Name) + "\t";

var bits = Attributes.AsAttributesArray();

for(int i = 0; i < 3; i++)

{

if (bits[i])

str += '1';

else

str += '0';

}

str += "\t";

bits = Rights.AsAttributesArray();

for (int i = 0; i < 6; i++)

{

if (bits[i])

str += '1';

else

str += '0';

}

str += "\t";

var date = DateTime.ToString();

var dateArr = date.Insert(date.Length - 4, ":").Split(":");

dateArr[0] = dateArr[0].Insert(dateArr[0].Length - 4, ".");

dateArr[0] = dateArr[0].Insert(dateArr[0].Length - 7, ".");

dateArr[1] = dateArr[1].Insert(2, ":");

str += dateArr[0] + " - ";

str += dateArr[1] + "\t";

str += Size;

return str;

}

}

internal class Path

{

public string DirectoryPath { get; set; }

public string Filename { get; set; }

}

}

Session.cs

using Chaos.Structures;

using Chaos.Tools;

using Chaos.Tools.Exceptions;

namespace Chaos.SystemComponents

{

internal class Session

{

public User User { get; set; }

public bool IsRoot { get; set; } = false;

public string CurrentPath { get; set; } = "";

public const string RootDirectory = "root\\";

public override string ToString()

{

return "[" + User?.Name + "]" + CurrentPath + ">";

}

public void RequestForAccess()

{

if (User?.Name != "root")

{

throw new Exception("Ошибка права доступа. Вы не являетесь администратором"); //!!!!!!!

}

Console.WriteLine("Введите пароль: ");

Console.Write(this);

if (User.Password != Console.ReadLine())

{

throw new Exception("Ошибка доступа. Пароль был указан неверно.");

}

}

public void isWritable(FileRecord file)

{

var rights = file.Rights.AsAttributesArray();

if (IsRoot)

return;

if (User == null)

throw new AccessException();

if (User.Id == file.UserId)

{

if (!rights[1])

throw new AccessException();

return;

}

if (!rights[4])

{

throw new AccessException();

}

}

public void isReadable(FileRecord file)

{

var rights = file.Rights.AsAttributesArray();

if (IsRoot)

return;

if (User == null)

throw new AccessException();

if (User?.Id == file.UserId)

{

if (!rights[0])

throw new AccessException();

return;

}

if (!rights[3])

{

throw new AccessException();

}

}

}

}

Superblock.cs

namespace Chaos.Structures

{

internal static class Superblock

{

public static string Title { get; private set; } = "ChaOS";

public static short ClusterSize { get; private set; } = 4096;

public static short FATSize { get; private set; } = 1024;

public static short FATCount { get; private set; } = 2;

public static short UsersAreaSize { get; private set; } = 176;

public static int DataAreaSize { get; private set; } = 2099392;

public static int Size { get; private set; } = 16;

public static int TablesStart => Size;

public static int UsersAreaStart => TablesStart + (FATSize \* FATCount);

public static int DataAreaStart => UsersAreaStart + UsersAreaSize;

public static int FSSize => 2099392;

}

}

User.cs

using Chaos.Tools;

using Chaos.Tools.Exceptions;

namespace Chaos.Structures

{

internal class User

{

public static int SizeInBytes => 22;

private string name;

private string pass;

public short Id { get; set; }

public string Name

{

get => name; set

{

if (value.Length < 10) name = value;

else throw new UserLengthException();

}

}

public string Password

{

get => pass; set

{

if (value.Length < 10) pass = value;

else throw new UserLengthException();

}

}

public static explicit operator byte[](User user)

{

var arr = new byte[22];

user.Id.ToBytes().CopyTo(arr, 0);

user.Name.ToBytes().CopyTo(arr, 2);

user.Password.ToBytes().CopyTo(arr, 12);

return arr;

}

public override string ToString()

{

return $"{Id}\t{Name}\t";

}

}

}

Converter.cs

using System.Collections;

using System.Text;

using Chaos.Structures;

namespace Chaos.Tools

{

internal static class Converter

{

public static byte[] ToBytes(this string str)

{

return Encoding.UTF8.GetBytes(str);

}

public static byte[] ToBytes(this int value)

{

return BitConverter.GetBytes(value);

}

public static byte[] ToBytes(this short value)

{

return BitConverter.GetBytes(value);

}

public static byte[] ToBytes(this long value)

{

return BitConverter.GetBytes(value);

}

public static byte ToByte(this BitArray bits)

{

byte[] arr = new byte[1];

for (int i = 0; i < 4; i++)

{

bool bit = bits[i];

bits[i] = bits[8 - i - 1];

bits[8 - i - 1] = bit;

}

bits.CopyTo(arr, 0);

return arr[0];

}

public static byte ToByte(this string str)

{

BitArray bits = new BitArray(8);

for (int i =0; i < str.Length; i++)

{

bits[7-i] = str[i] == '1';

}

byte[] bt= new byte[1];

bits.CopyTo(bt, 0);

return bt[0];

}

public static string ToStr(this byte[] arr)

{

return Encoding.UTF8.GetString(arr);

}

public static string Clear(this string str)

{

return str.Replace("\0", "");

}

public static int ToInt(this byte[] arr)

{

return BitConverter.ToInt32(arr);

}

public static short ToShort(this byte[] arr)

{

return BitConverter.ToInt16(arr);

}

public static long ToLong(this byte[] arr)

{

return BitConverter.ToInt64(arr);

}

public static long ToLong(this DateTime dateTime)

{

var date = dateTime.ToShortDateString().Replace(".", "");

if (date.Length == 7)

date = date.Insert(0, "0");

var time = dateTime.ToShortTimeString().Replace(".",

"").Replace(":", "");

if (time.Length == 3)

time = time.Insert(0, "0");

return long.Parse(date + time);

}

public static List<User> AsUserList(this byte[] data)

{

var list = new List<User>();

for (int i = 0; i < data.Length; i += User.SizeInBytes)

{

var id = new byte[2];

var name = new byte[10];

var password = new byte[10];

Array.Copy(data, i, id, 0, 2);

Array.Copy(data, i + 2, name, 0, 10);

Array.Copy(data, i + 12, password, 0, 10);

var user = new User

{

Id = id.ToShort(),

Name = name.ToStr().Clear(),

Password = password.ToStr().Clear()

};

if (user.Id != 0)

list.Add(user);

}

return list;

}

public static int GetLastCharIndex(this string str, char c)

{

var lastIndex = -1;

for (int i = 0; i < str.Length; i++)

{

if (str[i] == c)

lastIndex = i;

}

return lastIndex;

}

public static Structures.Path GetPath(this string str)

{

var index1 = str.GetLastCharIndex('\\');

var directoryStr = index1 != -1 ? str.Substring(0, index1) :

null;

var Filename = (index1 != -1 ? str.Remove(0, index1) : str).Trim('\\');

return new Structures.Path

{

DirectoryPath = directoryStr,

Filename = Filename

};

}

public static BitArray AsAttributesArray(this byte val)

{

var arr = new BitArray(new byte[] { val });

for(int i = 0; i < 4; i++)

{

bool bit = arr[i];

arr[i] = arr[8 - i - 1];

arr[8 - i - 1] = bit;

}

return arr;

}

public static List<FileRecord> AsFileList(this byte[] data)

{

var list = new List<FileRecord>();

for (int i = 0; i < data.Length; i += FileRecord.RecordSize)

{

if (data.Length - i < FileRecord.RecordSize)

return list;

var name = new byte[11];

var attr = new byte[1];

var rights = new byte[1];

var dateTime = new byte[8];

var cluster = new byte[2];

var size = new byte[4];

var userId = new byte[2];

Array.Copy(data, i, name, 0, 11);

Array.Copy(data, i + 11, attr, 0, 1);

Array.Copy(data, i + 15, rights, 0, 1);

Array.Copy(data, i + 16, dateTime, 0, 8);

Array.Copy(data, i + 24, cluster, 0, 2);

Array.Copy(data, i + 26, size, 0, 4);

Array.Copy(data, i + 30, userId, 0, 2);

var file = new FileRecord

{

Name = name.ToStr().Clear(),

Attributes = attr[0],

Rights = rights[0],

DateTime = dateTime.ToLong(),

Cluster = cluster.ToShort(),

Size = size.ToInt(),

UserId = userId.ToShort()

};

if (file.Name.Length != 0)

list.Add(file);

}

return list;

}

}

}

Exceptions.cs

namespace Chaos.Tools.Exceptions

{

internal class NoFileException :Exception

{

public override string Message => "Нет соответствующих объектов.";

}

internal class UserLengthException : Exception

{

public override string Message => "Длина имени пользователя и пароля не должна привышать 10 символов";

}

internal class UsernameException : Exception

{

public override string Message => "Ошибка! Пользователь с таким именем уже существует.";

}

internal class NoFreeSpaceException : Exception

{

public override string Message => "Не хватает свободного места для записи.";

}

internal class SessionLoginException : Exception

{

public override string Message => "Ошибка! Вы уже находитесь в сессии.";

}

internal class AccessException : Exception

{

public override string Message => "Доступ запрещен.";

}

internal class NoFreeClusterException : Exception

{

public override string Message => "Нет свободных кластеров.";

}

internal class FileExistException : Exception

{

public override string Message => "Файл с таким именем уже существует.";

}

internal class NotInSessionException : Exception

{

public override string Message => "Ошибка! Вы не находитесь в сессии.";

}

internal class FilenameException : Exception

{

public override string Message => "Длина имени файла не должна привышать 11 символов";

}

internal class ChmodException : Exception

{

public override string Message => "Неверное количество разрешений chmod";

}

internal class ChattrException : Exception

{

public override string Message => "Неверное количество атрибутов chattr";

}

internal class ReadOnlyException : Exception

{

public override string Message => "Файл имеет атрибут только для чтения";

}

}

StreamOperations.cs

namespace Chaos.Tools

{

internal class StreamOperations : IDisposable

{

private FileStream fstream;

public void Dispose()

{

fstream?.Close();

fstream?.Dispose();

}

public void Open(string filename)

{

fstream = new FileStream(filename, FileMode.OpenOrCreate, FileAccess.ReadWrite);

}

public async Task Write(byte[] buffer, long position)

{

fstream.Position = position;

await fstream.WriteAsync(buffer, 0, buffer.Length);

await fstream.FlushAsync();

}

public async Task Read(byte[] buffer, long position)

{

fstream.Position = position;

await fstream.ReadAsync(buffer, 0, buffer.Length);

}

}

}

SystemCommands.cs

using System.Text.RegularExpressions;

using Chaos.Structures;

using Chaos.Tools;

using Chaos.Tools.Exceptions;

using System.Collections;

namespace Chaos.SystemComponents.Commands

{

internal interface ICommand

{

public void Info();

public bool isExecutable(string command);

public Task Run(string command);

}

internal class DiskFormatCommand : ICommand

{

private readonly FileSystem fs;

private Session session;

public bool isExecutable(string command) => Regex.IsMatch(command, "format");

public DiskFormatCommand(FileSystem fs, Session session)

{

this.fs = fs;

this.session = session;

}

public async Task Run(string command)

{

session.RequestForAccess();

await fs.DiskFormat();

session.User = null;

session.CurrentPath = "";

session.IsRoot = false;

}

public void Info()

{

Console.WriteLine("format\t - форматирование файловой системы");

}

}

internal class AddUserCommand : ICommand

{

private readonly Session session;

private readonly FileSystem fs;

public AddUserCommand(FileSystem fs, Session session)

{

this.session = session;

this.fs = fs;

}

public bool isExecutable(string command) => Regex.IsMatch(command, "adduser [A-Za-z]+ [A-Za-z1-9]+");

public async Task Run(string command)

{

var arr = command.Split(' ');

session.RequestForAccess();

var usersData = await fs.ReadData(Superblock.UsersAreaStart, Superblock.UsersAreaSize);

var userList = usersData.AsUserList();

if (userList.FirstOrDefault(x => x.Name == arr[1]) != null)

throw new UsernameException();

var user = new User()

{

Id = GenerateID(userList),

Name = arr[1],

Password = arr[2]

};

var cluster = await fs.GetCluster();

var userDirectory = new FileRecord()

{

Name = user.Name,

Attributes = 160,

Rights = 224,

DateTime = DateTime.Now.ToLong(),

Cluster = cluster,

Size = 0,

UserId = user.Id

};

var recordcluster = await fs.FindLastCluster(1);

await fs.WriteData(fs.EOF.ToBytes(), Superblock.TablesStart + cluster \* 2);

await fs.WriteData(fs.EOF.ToBytes(), Superblock.TablesStart + Superblock.FATSize + cluster \* 2);

await fs.WriteData((byte[])user, Superblock.UsersAreaStart, Superblock.DataAreaStart);

await fs.WriteData((byte[])userDirectory, Superblock.DataAreaStart + recordcluster \* 2 - 2,

Superblock.DataAreaStart + recordcluster \* 2 - 2 + Superblock.ClusterSize);

Console.WriteLine("Пользователь успешно создан.");

}

public void Info()

{

Console.WriteLine("adduser [имя польз.] [пароль]\t - создание нового пользователя");

}

private short GenerateID(List<User> list)

{

short id = 2;

for (int i = 1; i < list.Count; i++)

{

if (id == list[i].Id)

id++;

}

return id;

}

}

internal class LogInCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

public bool isExecutable(string command) => Regex.IsMatch(command, "login [A-Za-z1-9]+ [A-Za-z1-9]+");

public LogInCommand(FileSystem fs, Session session)

{

this.fs = fs;

this.session = session;

}

public async Task Run(string command)

{

if (session.User != null)

throw new SessionLoginException();

var arr = command.Split(' ');

var data = await fs.ReadData(Superblock.UsersAreaStart, Superblock.UsersAreaSize);

var usersData = await fs.ReadData(Superblock.UsersAreaStart, Superblock.UsersAreaSize);

var userList = usersData.AsUserList();

for (int i = 0; i < userList.Count; i++)

{

if (userList[i].Name == arr[1] &&

userList[i].Password == arr[2])

{

session.User = new User()

{

Id = userList[i].Id,

Name = userList[i].Name,

Password = userList[i].Password

};

session.IsRoot = session.User.Name == "root";

session.CurrentPath = Session.RootDirectory;

Console.WriteLine("Авторизация выполнена успешно.");

return;

}

}

Console.WriteLine("Ошибка. Неверное имя пользователя или пароль");

}

public void Info()

{

Console.WriteLine("login [имя польз.] [пароль]\t - авторизация пользователя");

}

}

internal class LogOutCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

public LogOutCommand(FileSystem fs, Session session)

{

this.fs = fs;

this.session = session;

}

public bool isExecutable(string command) => Regex.IsMatch(command, "logout");

public async Task Run(string command)

{

if (session.User == null)

throw new NotInSessionException();

session.User = null;

session.CurrentPath = "";

session.IsRoot = false;

}

public void Info()

{

Console.WriteLine("logout\t - выход из учетной записи");

}

}

internal class CreateDirCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

public CreateDirCommand(FileSystem fs, Session

session)

{

this.fs = fs;

this.session = session;

}

public bool isExecutable(string command) =>

Regex.IsMatch(command, "crdir ([A-Za-zА-Яа-я1-9\\ ])+");

public async Task Run(string command)

{

if (session.User == null)

throw new AccessException();

var arr = command.Split(' ');

var path = arr[1].GetPath();

if (path.Filename.Length > 11)

throw new FilenameException();

var dir = await fs.GetDir(path.DirectoryPath ??

session.CurrentPath.Remove(session.CurrentPath.Length - 1));

session.isWritable(dir);

var list = await fs.GetAllFiles(dir.Cluster);

list.ForEach(x =>

{

if (x.Name == path.Filename)

throw new FileExistException();

});

var cluster = await fs.GetCluster();

var newDir = new FileRecord()

{

Name = path.Filename,

Attributes = 128,

Rights = 224,

DateTime = DateTime.Now.ToLong(),

Cluster = cluster,

Size = 0,

UserId = session.User.Id

};

var recordcluster = await fs.FindLastCluster(dir.Cluster);

await fs.WriteData(fs.EOF.ToBytes(), Superblock.TablesStart + cluster \* 2);

await fs.WriteData(fs.EOF.ToBytes(), Superblock.TablesStart + Superblock.FATSize + cluster \* 2);

await fs.WriteData((byte[])newDir, recordcluster \* Superblock.ClusterSize - Superblock.ClusterSize

+ Superblock.DataAreaStart, recordcluster \* Superblock.ClusterSize + Superblock.DataAreaStart);

}

public void Info()

{

Console.WriteLine("crdir [имя каталога]\t - создание нового каталога");

}

}

internal class OpenDirCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

public OpenDirCommand(FileSystem fs, Session session)

{

this.fs = fs;

this.session = session;

}

public bool isExecutable(string command) =>

(Regex.IsMatch(command, "opendir ([A-Za-zА-Яа-я1-9\\ ])+") ||

Regex.IsMatch(command, "opendir .."));

public async Task Run(string command)

{

var path = command.Split(' ')[1];

if (path == "..")

{

if (session.CurrentPath == Session.RootDirectory)

return;

int x = session.CurrentPath.Length - 1;

session.CurrentPath = session.CurrentPath.Remove(x);

var index = session.CurrentPath.GetLastCharIndex('\\');

session.CurrentPath = session.CurrentPath.Substring(0, index + 1);

}

else if (path.Contains('\\'))

{

await fs.GetDir(path);

session.CurrentPath = path;

}

else

{

if (session.CurrentPath == Session.RootDirectory && path == "root")

throw new NoFileException();

await fs.GetDir(session.CurrentPath + path);

session.CurrentPath = session.CurrentPath + path + "\\";

}

}

public void Info()

{

Console.WriteLine("opendir [имя каталога]\t - Переход в каталог");

}

}

internal class DirListCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

public DirListCommand(FileSystem fs, Session session)

{

this.fs = fs;

this.session = session;

}

public bool isExecutable(string command) =>

Regex.IsMatch(command, "dirlist( [/A-Za-z1-9]+)?");

public async Task Run(string command)

{

var arr = command.Split(' ');

FileRecord dir;

if (arr.Length == 1)

{

dir = await

fs.GetDir(session.CurrentPath.Remove(session.CurrentPath.Length - 1));

}

else

{

dir = await fs.GetDir(arr[1]);

}

var usersData = await fs.ReadData(Superblock.UsersAreaStart, Superblock.UsersAreaSize);

var userList = usersData.AsUserList();

var list = await fs.GetAllFiles(dir.Cluster);

Console.WriteLine("File\tUser\tAttr\tRights\tDate - Time\t\tSize");

list.ForEach(x =>

{

if (!x.Attributes.AsAttributesArray()[1])

Console.WriteLine(x.ToString(userList));

});

}

public void Info()

{

Console.WriteLine("dirlist [имя каталога]\t - вывод содержимого каталога");

}

}

internal class RenameCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

public RenameCommand(FileSystem fs, Session session)

{

this.fs = fs;

this.session = session;

}

public bool isExecutable(string command) =>

Regex.IsMatch(command, "rename ([A-Za-zА-Яа-я1-9])+ ([AZa-zА-Яа-я1-9])");

public async Task Run(string command)

{

if (session.User == null)

throw new AccessException();

var arr = command.Split(' ');

if (arr[1].Length > 11 || arr[2].Length > 11)

throw new FilenameException();

var oldName = arr[1];

var newName = arr[2];

var parentdir = await fs.GetDir(session.CurrentPath.Remove(session.CurrentPath.Length - 1));

session.isWritable(parentdir);

var info = await fs.FindFile(oldName, parentdir.Cluster) ??

throw new NoFileException();

if (((FileRecord)info[0]).Attributes.AsAttributesArray()[2])

throw new ReadOnlyException();

session.isWritable((FileRecord)info[0]);

var newNameBytes = new byte[11];

newName.ToBytes().CopyTo(newNameBytes, 0);

await fs.WriteData(newNameBytes, Superblock.DataAreaStart + ((short)info[1] - 1) \*

Superblock.ClusterSize + ((int)info[2]) \* FileRecord.RecordSize);

}

public void Info()

{

Console.WriteLine("rename [имя файла]\t - переименования файла/каталога");

}

}

internal class ChangeAttributesCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

public ChangeAttributesCommand(FileSystem fs, Session session)

{

this.fs = fs;

this.session = session;

}

public bool isExecutable(string command) => Regex.IsMatch(command, "chattr ([A-Za-z1-9\\ ])+ (0|1)(0|1)");

public async Task Run(string command)

{

if (session.User == null)

throw new AccessException();

var arr = command.Split(' ');

if (arr[2].Length != 2)

throw new ChattrException();

var path = arr[1].GetPath();

var dir = await fs.GetDir(path.DirectoryPath ??

session.CurrentPath.Remove(session.CurrentPath.Length - 1));

var info = await fs.FindFile(path.Filename, dir.Cluster) ??

throw new NoFileException();

session.isWritable((FileRecord)info[0]);

var attr = ((FileRecord)info[0]).Attributes.AsAttributesArray();

attr[1] = arr[2].Substring(0, 1) == "1";

attr[2] = arr[2].Substring(1, 1) == "1";

((FileRecord)info[0]).Attributes = attr.ToByte();

await fs.WriteData((byte[])(FileRecord)info[0], Superblock.DataAreaStart + ((short)info[1] - 1) \*

Superblock.ClusterSize + (int)info[2] \* FileRecord.RecordSize);

}

public void Info()

{

Console.WriteLine("chattr [имя файла] [права доступа]\t - задание новых разрешений пользователй");

}

}

internal class ChangeRightsCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

public ChangeRightsCommand(FileSystem fs,

Session session)

{

this.fs = fs;

this.session = session;

}

public bool isExecutable(string command) =>

Regex.IsMatch(command, "chmod ([A-Za-z1-9\\ ])+ (0|1)(0|1)(0|1)(0|1)(0|1)(0|1)");

public async Task Run(string command)

{

if (session.User == null)

throw new AccessException();

var arr = command.Split(' ');

if (arr[2].Length != 6)

throw new ChmodException();

var path = arr[1].GetPath();

var dir = await fs.GetDir(path.DirectoryPath ??

session.CurrentPath.Remove(session.CurrentPath.Length - 1));

var info = await fs.FindFile(path.Filename, dir.Cluster) ??

throw new NoFileException();

session.isWritable((FileRecord)info[0]);

((FileRecord)info[0]).Rights = arr[2].ToByte();

await fs.WriteData((byte[])(FileRecord)info[0], Superblock.DataAreaStart + ((short)info[1] - 1) \*

Superblock.ClusterSize + (int)info[2] \* FileRecord.RecordSize);

}

public void Info()

{

Console.WriteLine("chmod [имя файла] [права доступа]\t - задание новых разрешений пользователй");

}

}

internal class UserListCommand : ICommand

{

private readonly FileSystem fs;

public UserListCommand(FileSystem fs)

{

this.fs = fs;

}

public bool isExecutable(string command) => command == "userlist";

public async Task Run(string command)

{

var data = await fs.ReadData(Superblock.UsersAreaStart, Superblock.UsersAreaSize);

var list = data.AsUserList();

Console.WriteLine("ID\t Name");

list.ForEach(x => Console.WriteLine(x));

}

public void Info()

{

Console.WriteLine("userlist\t - информация о всех пользователях");

}

}

internal class DeleteUserCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

public DeleteUserCommand(FileSystem fs, Session session)

{

this.fs = fs;

this.session = session;

}

public bool isExecutable(string command) => Regex.IsMatch(command, "deluser [A-Za-z]+");

public async Task Run(string command)

{

var arr = command.Split(' ');

var user = arr[1];

if (user == "root") throw new AccessException();

session.RequestForAccess();

var data = await fs.ReadData(Superblock.UsersAreaStart, Superblock.UsersAreaSize);

var userlist = data.AsUserList();

for (int i = 0; i < userlist.Count; i++)

{

if (userlist[i].Name == user)

{

var info = await fs.FindFile(user, 1) ??

throw new NoFileException();

var newName = user;

if (newName.Length > 7)

newName.Remove(6);

newName += DateTime.Now.Date.ToString().Remove(5).Replace(".", "");

((FileRecord)info[0]).Name = newName;

((FileRecord)info[0]).DateTime = DateTime.Now.ToLong();

((FileRecord)info[0]).Attributes = 224;

((FileRecord)info[0]).UserId = 1;

await fs.WriteData((byte[])(FileRecord)info[0], Superblock.DataAreaStart + ((short)info[1] - 1) \*

Superblock.ClusterSize + (int)info[2] \* FileRecord.RecordSize);

await fs.WriteData(new byte[User.SizeInBytes], Superblock.UsersAreaStart + i \* User.SizeInBytes);

Console.WriteLine("Пользователь удален.");

return;

}

}

throw new Exception("Пользователь не найден.");

}

public void Info()

{

Console.WriteLine("deluser [имя польз.]\t delete users");

}

}

internal class CreateFileCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

public CreateFileCommand(FileSystem fs, Session session)

{

this.fs = fs;

this.session = session;

}

public bool isExecutable(string command) => Regex.IsMatch(command, "crfile ([A-Za-zА-Яа-я1-9\\ ])+");

public async Task Run(string command)

{

if (session.User == null)

throw new AccessException();

var arr = command.Split(' ');

var path = arr[1].GetPath();

var dir = await fs.GetDir(path.DirectoryPath ??

session.CurrentPath.Remove(session.CurrentPath.Length - 1));

session.isWritable(dir);

var list = await fs.GetAllFiles(dir.Cluster);

list.ForEach(x =>

{

if (x.Name == path.Filename)

throw new FileExistException();

});

var cluster = await fs.GetCluster();

var newFile = new FileRecord()

{

Name = path.Filename,

Attributes = 0,

Rights = 192,

DateTime = DateTime.Now.ToLong(),

Cluster = cluster,

Size = 0,

UserId = session.User.Id

};

var recordcluster = await fs.FindLastCluster(dir.Cluster);

await fs.WriteData(fs.EOF.ToBytes(), Superblock.TablesStart + cluster \* 2);

await fs.WriteData(fs.EOF.ToBytes(), Superblock.TablesStart + Superblock.FATSize + cluster \* 2);

await fs.WriteData((byte[])newFile, recordcluster \* Superblock.ClusterSize - Superblock.ClusterSize

+ Superblock.DataAreaStart, recordcluster \* Superblock.ClusterSize + Superblock.DataAreaStart);

await fs.ResizeDirectories(path.DirectoryPath, 32);

}

public void Info()

{

Console.WriteLine("crfile [имя/путь файла]\t - создание файла ");

}

}

internal class FileWriteCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

private ArrayList fileinfo;

public FileWriteCommand(FileSystem fs, Session session)

{

this.fs = fs;

this.session = session;

}

public bool isExecutable(string command) => Regex.IsMatch(command, "filewrite (-a |-r )([A-Za-zА-Яа-я1-9\\ ])+");

public async Task Run(string command)

{

if (session.User == null)

throw new AccessException();

var arr = command.Split(' ');

var path = arr[2].GetPath();

var dir = await fs.GetDir(path.DirectoryPath ?? session.CurrentPath.Remove(session.CurrentPath.Length - 1));

fileinfo = await fs.FindFile(path.Filename, dir.Cluster) ??

throw new NoFileException();

if (((FileRecord)fileinfo[0]).Attributes.AsAttributesArray()[0])

throw new NoFileException();

session.isWritable((FileRecord)fileinfo[0]);

var inputData = string.Empty;

Console.WriteLine("(Чтобы остановить ввод нажмите Num0)Ввод данных:");

inputData += Console.ReadLine();

while (Console.ReadKey().Key != ConsoleKey.NumPad0)

{

inputData += Console.ReadLine();

}

((FileRecord)fileinfo[0]).Size = 0;

if (arr[1] == "-r")

await Rewrite(inputData.ToBytes());

else

await Append(inputData.ToBytes());

await fs.WriteData(((FileRecord)fileinfo[0]).Size.ToBytes(), Superblock.DataAreaStart +

((short)fileinfo[1] - 1) \* Superblock.ClusterSize + ((int)fileinfo[2]) \* FileRecord.RecordSize + 26);

await fs.ResizeDirectories(path.DirectoryPath, ((FileRecord)fileinfo[0]).Size);

}

private async Task Rewrite(byte[] data)

{

var cluster = ((FileRecord)fileinfo[0]).Cluster;

var next = cluster;

do

{

cluster = next;

await fs.WriteData(new byte[Superblock.ClusterSize], Superblock.DataAreaSize + Superblock.ClusterSize \* cluster);

next = await fs.GetNextCluster(cluster);

await fs.WriteData(new byte[2], Superblock.TablesStart + (cluster) \* 2);

await fs.WriteData(new byte[2], Superblock.TablesStart + Superblock.FATSize + (cluster) \* 2);

} while (cluster != next);

await WriteData(((FileRecord)fileinfo[0]).Cluster, data);

}

private async Task Append(byte[] data)

{

var last = await fs.FindLastCluster(((FileRecord)fileinfo[0]).Cluster);

var oldDataBytes = await fs.ReadData(Superblock.DataAreaStart + Superblock.ClusterSize \* (last - 1), Superblock.ClusterSize);

var oldData = oldDataBytes.ToStr().Trim('\0').ToBytes();

var newData = new byte[oldData.Length + data.Length];

oldData.CopyTo(newData, 0);

data.CopyTo(newData, oldData.Length);

await WriteData(last, newData);

}

private async Task WriteData(int cluster, byte[] data)

{

var newCluster = -1;

for (int i = 0; i < data.Length; i += Superblock.ClusterSize)

{

if (newCluster != -1)

{

await fs.WriteData(newCluster.ToBytes(), Superblock.TablesStart + cluster \* 2);

await fs.WriteData(newCluster.ToBytes(), Superblock.TablesStart + Superblock.FATSize + cluster \* 2);

cluster = newCluster;

}

var partData = new byte[Superblock.ClusterSize];

var size = (data.Length - i) >= Superblock.ClusterSize ? Superblock.ClusterSize : data.Length - i;

Array.Copy(data, i, partData, 0, size);

var startPosition = Superblock.DataAreaStart + Superblock.ClusterSize \* (cluster - 1);

await fs.WriteData(partData, startPosition);

await fs.WriteData(fs.EOF.ToBytes(), Superblock.TablesStart + cluster \* 2);

await fs.WriteData(fs.EOF.ToBytes(), Superblock.TablesStart + Superblock.FATSize + cluster \* 2);

((FileRecord)fileinfo[0]).Size += size;

newCluster = await fs.GetCluster();

}

}

public void Info()

{

Console.WriteLine("filewrite [имя/путь файла]\t - запить в файл");

}

}

internal class FileReadCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

public FileReadCommand(FileSystem fs, Session session)

{

this.fs = fs;

this.session = session;

}

public bool isExecutable(string command) => Regex.IsMatch(command, "fileread [A-Za-zА-Яа-я1-9\\ ]+");

public async Task Run(string command)

{

if (session.User == null)

throw new AccessException();

var path = command.Split(' ')[1].GetPath();

var dir = await fs.GetDir(path.DirectoryPath ?? session.CurrentPath.Remove(session.CurrentPath.Length - 1));

var list = await fs.GetAllFiles(dir.Cluster);

var file = list.Find(x => x.Name == path.Filename &&

!x.Attributes.AsAttributesArray()[0]) ??

throw new NoFileException();

session.isReadable(file);

var startcluster = file.Cluster;

var nextcluster = startcluster;

do

{

startcluster = nextcluster;

var fileData = await fs.ReadData(Superblock.DataAreaStart + Superblock.ClusterSize \* (startcluster - 1), Superblock.ClusterSize);

Console.Write(fileData.ToStr());

nextcluster = await fs.GetNextCluster(startcluster);

} while (startcluster != nextcluster);

Console.WriteLine();

}

public void Info()

{

Console.WriteLine("fileread [имя/путь файла]\t - Вывод содержимого файла");

}

}

internal class RemoveFileCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

private Task rmTask;

public RemoveFileCommand(FileSystem fs, Session session)

{

this.fs = fs;

this.session = session;

}

public bool isExecutable(string command) => Regex.IsMatch(command, "rmfile ([A-Za-z1-9\\ ]+)");

public async Task Run(string command)

{

if (session.User == null)

throw new AccessException();

var pathStr = command.Split(' ')[1];

var path = pathStr.GetPath();

var dir = await fs.GetDir(path.DirectoryPath ?? session.CurrentPath.Remove(session.CurrentPath.Length - 1));

var info =await fs.FindFile(path.Filename,dir.Cluster) ??

throw new NoFileException();

if (((FileRecord)info[0]).Attributes.AsAttributesArray()[0])

throw new NoFileException();

if (((FileRecord)info[0]).Attributes.AsAttributesArray()[2])

throw new ReadOnlyException();

session.isWritable(dir);

session.isWritable((FileRecord)info[0]);

await fs.WriteData(new byte[FileRecord.RecordSize], Superblock.DataAreaStart +

((short)info[1] - 1) \* Superblock.ClusterSize + ((int)info[2]) \* FileRecord.RecordSize);

var cluster = ((FileRecord)info[0]).Cluster;

var next = cluster;

do

{

cluster = next;

next = await fs.GetNextCluster(cluster);

await fs.WriteData(new byte[2], Superblock.TablesStart + cluster \* 2);

await fs.WriteData(new byte[2], Superblock.TablesStart + Superblock.FATSize + cluster \* 2);

await fs.WriteData(new byte[Superblock.ClusterSize], Superblock.DataAreaStart + (cluster-1) \* Superblock.ClusterSize);

} while (cluster != next);

await fs.ResizeDirectories(path.DirectoryPath, -32 - ((FileRecord)info[0]).Size);

}

public void Info()

{

Console.WriteLine("rmfile [имя/путь файла]\t - удаление файла");

}

}

internal class RemoveDirCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

public RemoveDirCommand(FileSystem fs, Session session)

{

this.fs = fs;

this.session = session;

}

public bool isExecutable(string command) => Regex.IsMatch(command, "rmdir ([A-Za-zА-Яа-я1-9\\ ])+");

public async Task Run(string command)

{

if (session.User == null)

throw new AccessException();

var pathStr = command.Split(' ')[1];

var path = pathStr.GetPath();

var dir = await fs.GetDir(path.DirectoryPath ?? session.CurrentPath.Remove(session.CurrentPath.Length - 1));

session.isWritable(dir);

//Считываем файл из диска

var info = await fs.FindFile(path.Filename, dir.Cluster) ??

throw new NoFileException();

if (!((FileRecord)info[0]).Attributes.AsAttributesArray()[0])

throw new NoFileException();

if (((FileRecord)info[0]).Attributes.AsAttributesArray()[2])

throw new ReadOnlyException();

session.isWritable((FileRecord)info[0]);

await RecursRemoval((FileRecord)info[0]);

await fs.WriteData(new byte[FileRecord.RecordSize], Superblock.DataAreaStart +

((short)info[1] - 1) \* Superblock.ClusterSize + ((int)info[2]) \* FileRecord.RecordSize);

await fs.ResizeDirectories(path.DirectoryPath, -((FileRecord)info[0]).Size);

}

public void Info()

{

Console.WriteLine("rmdir [имя/путь каталога]\t - удалить каталог");

}

private async Task RecursRemoval(FileRecord fl)

{

var cluster = fl.Cluster;

var list = await fs.GetAllFiles(cluster);

var next = cluster;

do

{

cluster = next;

next = await fs.GetNextCluster(cluster);

await fs.WriteData(new byte[2], Superblock.TablesStart + cluster \* 2);

await fs.WriteData(new byte[2], Superblock.TablesStart + Superblock.FATSize + cluster \* 2);

await fs.WriteData(new byte[Superblock.ClusterSize], Superblock.DataAreaStart + (cluster - 1) \* Superblock.ClusterSize);

} while (cluster != next);

if (fl.Attributes.AsAttributesArray()[0])

{

for(int i=0; i < list.Count; i++)

await RecursRemoval(list[i]);

}

}

}

internal class MoveCommand : ICommand

{

private readonly FileSystem fs;

private readonly Session session;

public MoveCommand(FileSystem fs, Session session)

{

this.fs = fs;

this.session = session;

}

public bool isExecutable(string command) =>

Regex.IsMatch(command, "move ([A-Za-z1-9\\ ]+)([A-Za-z1-9\\ ]+)");

public async Task Run(string command)

{

if (session.User == null)

throw new AccessException();

var arr = command.Split(' ');

var path1Str = arr[1];

var path2Str = arr[2];

var path1 = path1Str.GetPath();

var path2 = path2Str.GetPath();

//Проверяем правильность указанных путей

var dir1 = await fs.GetDir(path1.DirectoryPath ?? session.CurrentPath.Remove(session.CurrentPath.Length - 1));

var dir2 = await fs.GetDir(path2.DirectoryPath ?? session.CurrentPath.Remove(session.CurrentPath.Length - 1));

session.isWritable(dir1);

session.isReadable(dir1);

session.isWritable(dir2);

var info = await fs.FindFile(path1.Filename, dir1.Cluster) ??

throw new NoFileException();

if (((FileRecord)info[0]).Attributes.AsAttributesArray()[2])

throw new ReadOnlyException();

session.isWritable((FileRecord)info[0]);

await fs.WriteData(new byte[FileRecord.RecordSize], Superblock.DataAreaStart +

((short)info[1] - 1) \* Superblock.ClusterSize + ((int)info[2]) \* FileRecord.RecordSize);

var cluster = await fs.FindLastCluster(dir2.Cluster);

await fs.WriteData((byte[])(FileRecord)info[0], Superblock.DataAreaStart + (cluster-1) \*

Superblock.ClusterSize, Superblock.DataAreaStart + (cluster - 1) \* Superblock.ClusterSize + Superblock.ClusterSize);

}

public void Info()

{

Console.WriteLine("move [имя/путь файла] [путь каталога]\t - переместить файл в указаный путь");

}

}

}

Межпроцессное взаимодействие

#include <sys/sem.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <string>

#include <iostream>

#include <unistd.h>

using namespace std;

class Semaphore

{

private:

struct sembuf sop\_lock[2] = {

{0, 0, 0},

{0, 1, 0}};

struct sembuf sop\_unlock[1] = {{0, -1, 0}};

int key;

int semid=0, perm = S\_IRUSR | S\_IWUSR | S\_IXUSR;

public:

Semaphore(int x){

semid=x;

}

void Create(){

key=semid+1;

semget(key, 1, IPC\_CREAT | perm);

}

void Lock()

{

semop(semid, &sop\_lock[0], 2);

cout<<"Процесс занял семафор"<<endl;

}

void Unlock()

{

semop(semid, &sop\_unlock[0], 1);

cout<<"Процесс освободил семафор"<<endl;

}

void Delete()

{

semctl(semid, 0, IPC\_RMID, 0);

}

};

class SharedMemory

{

private:

struct shmid\_ds shmbuffer;

char\* shared\_memory;

int id;

int perm = S\_IRUSR | S\_IWUSR | S\_IXUSR;

public:

SharedMemory(){}

SharedMemory(int key,int size=255){

id = shmget(key,size,IPC\_CREAT|perm);

printf("Id= %d",id);

cout<<endl;

}

void SetId(int x){

id=x;

}

void Attach(){

shared\_memory=(char\*)shmat(id, 0,0);

}

void Detach(){

shmdt(shared\_memory);

}

void Write(int i){

Attach();

sprintf(shared\_memory,"%d",i);

Detach();

printf("Записано: %d",i);

cout<<endl;

}

void Write(string s){

Attach();

sprintf(shared\_memory,"%s",s.c\_str());

Detach();

cout<<endl;

}

void Read(){

Attach();

printf("Считано: %s", shared\_memory);

Detach();

cout<<endl<<endl;

}

void Free(){

shmctl(id, IPC\_RMID, 0);

}

};

class Process

{

SharedMemory shmemory;

Semaphore semaph=Semaphore(0);

public:

~Process(){

semaph.Delete();

shmemory.Detach();

shmemory.Free();

}

void CreateSharedArea(int key){

semaph.Create();

shmemory = SharedMemory(key);

}

void SetId(int id){

shmemory.SetId(id);

}

void StartAutoInteraction(int count,int type)

{

for(int i = 0; i<count; i++){

semaph.Lock();

shmemory.Read();

shmemory.Write(i\*type);

semaph.Unlock();

usleep(100000);

}

}

};

int main()

{

string command;

Process proc{};

while (true)

{

cout<<">";

cin >> command;

if (command == "create")

{

int key;

cout<<"Ключ для области памяти:";

cin>>key;

proc.CreateSharedArea(key);

}

else if(command == "setid"){

int id;

cin>>id;

proc.SetId(id);

}

else if(command == "start"){

int count;

cout<<"Количество сообщений:";

cin>>count;

cout<<"Множитель:";

int type;

cin>>type;

proc.StartAutoInteraction(count,type);

}

else if (command == "help")

{

cout<<"create - создание разделяемой области памяти"<<endl;

cout<<"setid - задание id используемой области памяти"<<endl;

cout<<"start - запуск автоматического взаимодействия между процессами"<<endl;

cout<<"exit - выход"<<endl;

}

else if (command == "exit"){

proc.~Process();

return 0;

}else

continue;