ДИСЦИПЛИНА **Конфигурационное управление**

(полное наименование дисциплины без сокращений)

ИНСТИТУТ **Информационных технологий**

КАФЕДРА **Корпоративных информационных систем**

(полное наименование кафедры)

ВИД УЧЕБНОГО МАТЕРИАЛА **Задание для текущего контроля**

(в соответствии с пп.1-11)

ПРЕПОДАВАТЕЛЬ **П.Н. Советов**

(фамилия, имя, отчество)

СЕМЕСТР **3 семестр (осенний) 2025/2026 учебного года**

(указать семестр обучения, учебный год)

Конфигурационное управление

Сборник практических работ №1

ИКБО-20-24

РТУ МИРЭА – 2025

Оглавление

О практических работах

Практические работы (ПР) состоят из нескольких этапов. ПР выполняются очно и защита каждого этапа происходит на семинарских занятиях. Этапы работы над ПР сохраняются в публично доступном git-репозитории. Каждый этап разработки ПР должен быть отражен в истории коммитов с детальными сообщениями. Студент самостоятельно выбирает язык реализации.

Документация по ПР оформляется в виде readme.md, который содержит:

1. Общее описание.
2. Описание всех функций и настроек.
3. Описание команд для сборки проекта и запуска тестов.
4. Примеры использования.

Список публичных git-сервисов для репозиториев ПР:

1. github.com
2. gitea.com
3. gitlab.com
4. gitflic.ru
5. hub.mos.ru
6. gitverse.ru
7. gitee.com

Вариант №1

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является JSON-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
4. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `cal`, `cat`, `find`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `mkdir`, `mv`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №2

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `Эмулятор - [username@hostname]`.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.

2. Стартовый скрипт для выполнения команд эмулятора: поддерживает комментарии (используйте синтаксис из вашего языка реализации). При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является CSV-файл. Для двоичных данных используется base64 или аналогичный формат. Необходимо разобраться, как представлять вложенные элементы VFS.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `find`, `rev`, `who`.

3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `mkdir`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №3

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `Эмулятор - [username@hostname]`.
3. Реализовать парсер, который корректно обрабатывает аргументы в кавычках.
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта

на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является XML-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `clear`, `find`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.

4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `rm`, `cp`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №4

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать парсер, который поддерживает раскрытие переменных окружения реальной ОС (например, `$HOME`).
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: поддерживает комментарии (используйте синтаксис из вашего языка реализации). При

выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является CSV-файл. Для двоичных данных используется base64 или аналогичный формат. Необходимо разобраться, как представлять вложенные элементы VFS.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Необходимо реализовать команду `vfs-save путь` для сохранения состояния VFS на диск в исходном формате.
5. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
6. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `rev`, `head`, `tail`.

3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `mkdir`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №5

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме консольного интерфейса (CLI).
2. Приглашение к вводу должно формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `username@hostname:~$`.
3. Реализовать парсер, который поддерживает раскрытие переменных окружения реальной ОС (например, `$HOME`).
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта

на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является ZIP-архив. Для двоичных данных используется base64 или аналогичный формат.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
4. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `du`, `rev`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.

4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `chown`, `cp`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №6

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать парсер, который корректно обрабатывает аргументы в кавычках.
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: останавливается при первой ошибке. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является ZIP-архив. Для двоичных данных используется base64 или аналогичный формат.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `rev`, `pwd`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `touch`, `rm`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №7

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к лог-файлу.
   - Путь к стартовому скрипту.
2. Реализовать логирование событий вызова команд в файле формата XML. Событие лога дополнительно содержит:
   - Дата и время события.

3. Стартовый скрипт для выполнения команд эмулятора: останавливается при первой ошибке. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
4. Сообщить об ошибке во время исполнения стартового скрипта.
5. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является CSV-файл. Для двоичных данных используется base64 или аналогичный формат. Необходимо разобраться, как представлять вложенные элементы VFS.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Необходимо выдать информацию о загруженной VFS по служебной команде `vfs-info` (имя VFS и хеш SHA-256 ее данных).
5. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
6. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `cat`, `rev`, `whoami`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `chmod`, `rm`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №8

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: поддерживает комментарии (используйте синтаксис из вашего языка реализации). При

выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является JSON-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `whoami`, `tree`, `echo`.

3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `mkdir`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №9

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать парсер, который корректно обрабатывает аргументы в кавычках.
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является CSV-файл. Для двоичных данных используется base64 или аналогичный формат. Необходимо разобраться, как представлять вложенные элементы VFS.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `who`, `cat`, `tac`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.

4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `mv`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №10

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `Эмулятор - [username@hostname]`.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к лог-файлу.
   - Путь к стартовому скрипту.
2. Реализовать логирование событий вызова команд в файле формата XML. Событие лога дополнительно содержит:

– Дата и время события.

3. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
4. Сообщить об ошибке во время исполнения стартового скрипта.
5. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является директория на диске пользователя.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `whoami`, `history`.

3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `touch`.
2. По команде `help` выдать список команд с описанием их работы.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №11

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме консольного интерфейса (CLI).
2. Приглашение к вводу должно формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `username@hostname:~$`.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.

2. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является ZIP-архив. Для двоичных данных используется base64 или аналогичный формат.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `tac`, `rev`, `uptime`.

3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `rm`, `rmdir`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №12

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать парсер, который поддерживает раскрытие переменных окружения реальной ОС (например, `$HOME`).
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: поддерживает комментарии (используйте синтаксис из вашего языка реализации). При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является CSV-файл. Для двоичных данных используется base64 или аналогичный формат. Необходимо разобраться, как представлять вложенные элементы VFS.
3. Реализовать команду `vfs-init`, которая заменяет текущую VFS на VFS по умолчанию. В этом случае очищается и физическое представление VFS.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `date`, `history`, `clear`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.

4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `cp`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №13

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `Эмулятор - [username@hostname]`.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.

2. Стартовый скрипт для выполнения команд эмулятора: поддерживает комментарии (используйте синтаксис из вашего языка реализации). При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является XML-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Необходимо реализовать команду `vfs-save путь` для сохранения состояния VFS на диск в исходном формате.
5. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
6. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `uptime`, `history`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `touch`, `mv`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №14

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `Эмулятор - [username@hostname]`.
3. Реализовать парсер, который корректно обрабатывает аргументы в кавычках.
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта

на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является JSON-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Если путь к VFS при загрузке эмулятора не указан, то создать VFS по умолчанию в памяти.
5. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
6. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `wc`, `uniq`.

3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `mv`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №15

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме консольного интерфейса (CLI).
2. Приглашение к вводу должно формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `username@hostname:~$`.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.

2. Стартовый скрипт для выполнения команд эмулятора: останавливается при первой ошибке. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является CSV-файл. Для двоичных данных используется base64 или аналогичный формат. Необходимо разобраться, как представлять вложенные элементы VFS.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Необходимо выдать информацию о загруженной VFS по служебной команде `vfs-info` (имя VFS и хеш SHA-256 ее данных).
5. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
6. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `date`, `clear`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `rmdir`.
2. По команде `help` выдать список команд с описанием их работы.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №16

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать парсер, который поддерживает раскрытие переменных окружения реальной ОС (например, `$HOME`).
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является XML-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Если путь к VFS при загрузке эмулятора не указан, то создать VFS по умолчанию в памяти.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `clear`, `uptime`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.

4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `touch`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №17

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме консольного интерфейса (CLI).
2. Приглашение к вводу должно формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `username@hostname:~$`.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: поддерживает комментарии (используйте синтаксис из вашего языка реализации). При

выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является JSON-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
4. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `uniq`, `cat`, `tree`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.

4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `rm`, `mv`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №18

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме консольного интерфейса (CLI).
2. Приглашение к вводу должно содержать имя VFS.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Реализовать вывод параметров эмулятора в формате ключ-значение с помощью служебной команды `conf-dump`.
5. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является JSON-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
4. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `rev`, `echo`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.

4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `rm`, `mkdir`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №19

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме консольного интерфейса (CLI).
2. Приглашение к вводу должно содержать имя VFS.
3. Реализовать парсер, который поддерживает раскрытие переменных окружения реальной ОС (например, `$HOME`).
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: поддерживает комментарии (используйте синтаксис из вашего языка реализации). При

выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является CSV-файл. Для двоичных данных используется base64 или аналогичный формат. Необходимо разобраться, как представлять вложенные элементы VFS.
3. Если путь к VFS при загрузке эмулятора не указан, то создать VFS по умолчанию в памяти.
4. Реализовать команду `vfs-init`, которая заменяет текущую VFS на VFS по умолчанию. В этом случае очищается и физическое представление VFS.
5. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
6. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `clear`, `rev`, `tac`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `chown`, `cp`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №20

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать парсер, который корректно обрабатывает аргументы в кавычках.
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: останавливается при первой ошибке. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является XML-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Необходимо реализовать команду `vfs-save путь` для сохранения состояния VFS на диск в исходном формате.
5. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
6. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `tail`, `find`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.

4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `touch`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №21

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `Эмулятор - [username@hostname]`.
3. Реализовать парсер, который поддерживает раскрытие переменных окружения реальной ОС (например, `$HOME`).
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта

на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является ZIP-архив. Для двоичных данных используется base64 или аналогичный формат.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
4. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `rev`, `cal`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.

4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `mv`, `chown`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №22

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `Эмулятор - [username@hostname]`.
3. Реализовать парсер, который поддерживает раскрытие переменных окружения реальной ОС (например, `$HOME`).
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.

2. Стартовый скрипт для выполнения команд эмулятора: останавливается при первой ошибке. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
3. Реализовать вывод параметров эмулятора в формате ключ-значение с помощью служебной команды `conf-dump`.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является ZIP-архив. Для двоичных данных используется base64 или аналогичный формат.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `find`, `clear`, `tac`.

3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `touch`, `rm`.
2. Необходимо реализовать команду `vfs-load путь` для загрузки новой VFS с диска.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №23

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме консольного интерфейса (CLI).
2. Приглашение к вводу должно формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `username@hostname:~$`.
3. Реализовать парсер, который поддерживает раскрытие переменных окружения реальной ОС (например, `$HOME`).
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта

на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является JSON-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Поддержать вывод сообщения из файла `motd` при старте. Если `motd` существует, то он находится в корне VFS.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `find`, `history`.

3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `chown`, `rm`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №24

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `Эмулятор - [username@hostname]`.
3. Реализовать парсер, который корректно обрабатывает аргументы в кавычках.
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.

2. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является JSON-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Поддержать вывод сообщения из файла `motd` при старте. Если `motd` существует, то он находится в корне VFS.
5. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
6. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.

2. Реализовать новые команды: `date`, `rev`, `clear`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `touch`, `chown`.
2. Необходимо реализовать команду `vfs-load путь` для загрузки новой VFS с диска.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №25

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать парсер, который поддерживает раскрытие переменных окружения реальной ОС (например, `$HOME`).
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к лог-файлу.
   - Путь к стартовому скрипту.

2. Реализовать логирование событий вызова команд в файле формата JSON. Событие лога дополнительно содержит:
   - Дата и время события.
   - Сообщение о возникшей ошибке.
3. Стартовый скрипт для выполнения команд эмулятора: поддерживает комментарии (используйте синтаксис из вашего языка реализации). При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
4. Сообщить об ошибке во время исполнения стартового скрипта.
5. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является XML-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `du`, `find`, `pwd`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `mv`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №26

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме консольного интерфейса (CLI).
2. Приглашение к вводу должно формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `username@hostname:~$`.
3. Реализовать парсер, который корректно обрабатывает аргументы в кавычках.
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.

2. Стартовый скрипт для выполнения команд эмулятора: останавливается при первой ошибке. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является директория на диске пользователя.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
4. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `history`, `wc`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.

4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `cp`, `chmod`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №27

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме консольного интерфейса (CLI).
2. Приглашение к вводу должно формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `username@hostname:~$`.
3. Реализовать парсер, который корректно обрабатывает аргументы в кавычках.
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.

2. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является директория на диске пользователя.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `whoami`, `head`.

3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `mkdir`.
2. Необходимо реализовать команду `vfs-load путь` для загрузки новой VFS с диска.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №28

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме консольного интерфейса (CLI).
2. Приглашение к вводу должно содержать имя VFS.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.

2. Стартовый скрипт для выполнения команд эмулятора: останавливается при первой ошибке. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является CSV-файл. Для двоичных данных используется base64 или аналогичный формат. Необходимо разобраться, как представлять вложенные элементы VFS.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
4. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `cat`, `rev`, `whoami`.

3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `mv`, `rmdir`.
2. По команде `help` выдать список команд с описанием их работы.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №29

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `Эмулятор - [username@hostname]`.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.

2. Стартовый скрипт для выполнения команд эмулятора: останавливается при первой ошибке. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является директория на диске пользователя.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Если путь к VFS при загрузке эмулятора не указан, то создать VFS по умолчанию в памяти.
5. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
6. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `date`, `head`, `history`.

3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `cp`, `touch`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №30

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: поддерживает комментарии (используйте синтаксис из вашего языка реализации). При

выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является ZIP-архив. Для двоичных данных используется base64 или аналогичный формат.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Необходимо выдать информацию о загруженной VFS по служебной команде `vfs-info` (имя VFS и хеш SHA-256 ее данных).
5. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
6. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `tail`, `head`, `uniq`.

3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `cp`, `chown`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №31

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать парсер, который поддерживает раскрытие переменных окружения реальной ОС (например, `$HOME`).
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: поддерживает комментарии (используйте синтаксис из вашего языка реализации). При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является ZIP-архив. Для двоичных данных используется base64 или аналогичный формат.
3. Необходимо выдать информацию о загруженной VFS по служебной команде `vfs-info` (имя VFS и хеш SHA-256 ее данных).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `uptime`, `cal`, `whoami`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.

4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `chown`, `rm`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №32

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме консольного интерфейса (CLI).
2. Приглашение к вводу должно содержать имя VFS.
3. Реализовать парсер, который корректно обрабатывает аргументы в кавычках.
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: поддерживает комментарии (используйте синтаксис из вашего языка реализации). При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является директория на диске пользователя.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `pwd`, `cal`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `rmdir`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №33

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать парсер, который корректно обрабатывает аргументы в кавычках.
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.

2. Стартовый скрипт для выполнения команд эмулятора: останавливается при первой ошибке. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является XML-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `uniq`, `whoami`, `history`.

3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `cp`, `rm`.
2. Необходимо реализовать команду `vfs-load путь` для загрузки новой VFS с диска.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №34

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме консольного интерфейса (CLI).
2. Приглашение к вводу должно содержать имя VFS.
3. Реализовать парсер, который поддерживает раскрытие переменных окружения реальной ОС (например, `$HOME`).
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта

на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является директория на диске пользователя.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `whoami`, `history`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.

4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `cp`, `mkdir`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №35

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `Эмулятор - [username@hostname]`.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.

2. Стартовый скрипт для выполнения команд эмулятора: выполняет команды последовательно, ошибочные строки пропускает. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является JSON-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
4. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `head`, `tac`.

3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `mkdir`.
2. Необходимо реализовать команду `vfs-load путь` для загрузки новой VFS с диска.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №36

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме консольного интерфейса (CLI).
2. Приглашение к вводу должно формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `username@hostname:~$`.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
   - Путь к конфигурационному файлу.
2. Конфигурационный файл в формате XML содержит:
   - Путь к физическому расположению VFS.

– Путь к стартовому скрипту.

3. Приложение считывает параметры из командной строки и конфигурационного файла.
4. Логика приоритетов: значения из файла имеют приоритет над значениями из командной строки.
5. Стартовый скрипт для выполнения команд эмулятора: поддерживает комментарии (используйте синтаксис из вашего языка реализации). При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
6. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является CSV-файл. Для двоичных данных используется base64 или аналогичный формат. Необходимо разобраться, как представлять вложенные элементы VFS.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
4. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `tac`, `uptime`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `touch`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №37

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: поддерживает комментарии (используйте синтаксис из вашего языка реализации). При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является XML-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
4. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `cat`, `wc`, `tail`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `rm`, `mkdir`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №38

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен содержать имя VFS.
3. Реализовать парсер, который корректно обрабатывает аргументы в кавычках.
4. Сообщить об ошибке выполнения команд (неизвестная команда, неверные аргументы).
5. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
6. Реализовать команду `exit`.
7. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
8. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Пользовательское приглашение к вводу, которое будет отображаться в REPL.
   - Путь к стартовому скрипту.

- Путь к конфигурационному файлу.

2. Конфигурационный файл в формате JSON содержит:
   - Путь к физическому расположению VFS.
   - Пользовательское приглашение к вводу, которое будет отображаться в REPL.
   - Путь к стартовому скрипту.
3. Приложение считывает параметры из командной строки и конфигурационного файла.
4. Логика приоритетов: значения из файла имеют приоритет над значениями из командной строки.
5. Сообщить об ошибке чтения конфигурационного файла.
6. Стартовый скрипт для выполнения команд эмулятора: останавливается при первой ошибке. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
7. Сообщить об ошибке во время исполнения стартового скрипта.
8. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
9. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является CSV-файл. Для двоичных данных используется base64 или аналогичный формат. Необходимо разобраться, как представлять вложенные элементы VFS.
3. Сообщить об ошибке загрузки VFS (файл не найден, неверный формат).
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с

различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).

5. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `history`, `cat`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `mv`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №39

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме графического интерфейса (GUI).
2. Заголовок окна должен формироваться на основе реальных данных ОС, в которой исполняется эмулятор. Пример: `Эмулятор - [username@hostname]`.
3. Реализовать парсер, который поддерживает раскрытие переменных окружения реальной ОС (например, `$HOME`).
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
2. Стартовый скрипт для выполнения команд эмулятора: останавливается при первой ошибке. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.

3. Сообщить об ошибке во время исполнения стартового скрипта.
4. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является CSV-файл. Для двоичных данных используется base64 или аналогичный формат. Необходимо разобраться, как представлять вложенные элементы VFS.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
4. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.
2. Реализовать новые команды: `whoami`, `echo`, `date`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.

4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `mv`, `chown`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Вариант №40

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на работу в командной строке UNIX-подобной ОС.

Этап 1. REPL

Цель: создать минимальный прототип. Большинство функций в нем пока представляют собой заглушки, но диалог с пользователем уже поддерживается.

Требования:

1. Приложение должно быть реализовано в форме консольного интерфейса (CLI).
2. Приглашение к вводу должно содержать имя VFS.
3. Реализовать простой парсер, который разделяет ввод на команду и аргументы по пробелам.
4. Реализовать команды-заглушки, которые выводят свое имя и аргументы: `ls`, `cd`.
5. Реализовать команду `exit`.
6. Продемонстрировать работу прототипа в интерактивном режиме. Необходимо показать примеры работы всей реализованной функциональности, включая обработку ошибок.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Конфигурация

Цель: сделать эмулятор настраиваемым, то есть поддержать ввод параметров пользователя в приложение. Организовать для этого этапа отладочный вывод всех заданных параметров при запуске эмулятора.

Требования:

1. Параметры командной строки:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.
   - Путь к конфигурационному файлу.
2. Конфигурационный файл в формате JSON содержит:
   - Путь к физическому расположению VFS.
   - Путь к стартовому скрипту.

3. Приложение считывает параметры из командной строки и конфигурационного файла.
4. Логика приоритетов: значения из файла имеют приоритет над значениями из командной строки.
5. Стартовый скрипт для выполнения команд эмулятора: останавливается при первой ошибке. При выполнении скрипта на экране отображается как ввод, так и вывод, имитируя диалог с пользователем.
6. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования всех поддерживаемых параметров командной строки.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. VFS

Цель: подключить виртуальную файловую систему (VFS).

Требования:

1. Все операции должны производиться в памяти. Запрещается распаковывать или иным образом модифицировать данные VFS, за исключением возможных служебных команд.
2. Источником VFS является JSON-файл. Для двоичных данных используется base64 или аналогичный формат.
3. Создать несколько скриптов реальной ОС, в которой выполняется эмулятор. Включить в каждый скрипт вызовы эмулятора для тестирования работы с различными вариантами VFS (минимальный, несколько файлов, не менее 3 уровней файлов и папок).
4. Создать стартовый скрипт для тестирования всех реализованных на этом и прошлых этапах команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
5. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Основные команды

Цель: поддержать команды, имитирующие работу в UNIX-подобной командной строке.

Требования:

1. Необходимо реализовать логику для `ls` и `cd`.

2. Реализовать новые команды: `who`, `history`, `rev`.
3. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Дополнительные команды

Цель: поддержать более сложные команды, изменяющие состояние VFS, при этом модификации должны осуществляться только в памяти.

Требования:

1. Реализовать команды: `rmdir`.
2. Создать стартовый скрипт для тестирования всех реализованных на этом этапе команд. Добавить туда примеры всех режимов команд, включая работу с VFS и обработку ошибок.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.