Лабораторная работа №12

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Легиньких Галина Андреевна НФИбд-02-21

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Теоретическое введение

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:

– *оболочка Борна (Bourne shell или sh)* — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;

– *C-оболочка (или csh)* — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;

– *оболочка Корна (или ksh)* — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;

– *BASH* — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

3 Выполнение работы

1. Написала командный файл, реализующий упрощённый механизм семафоров. Командный файл в течение некоторого времени t1 дожидается освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использует его в течение некоторого времени t2<>t1, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустила командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (> /dev/tty#, где # — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработала программу так, чтобы имелась возможность взаимодействия трёх и более процессов.(рис. 3.1)(рис. 3.2)

Рис. 3.1: Скрипт 1

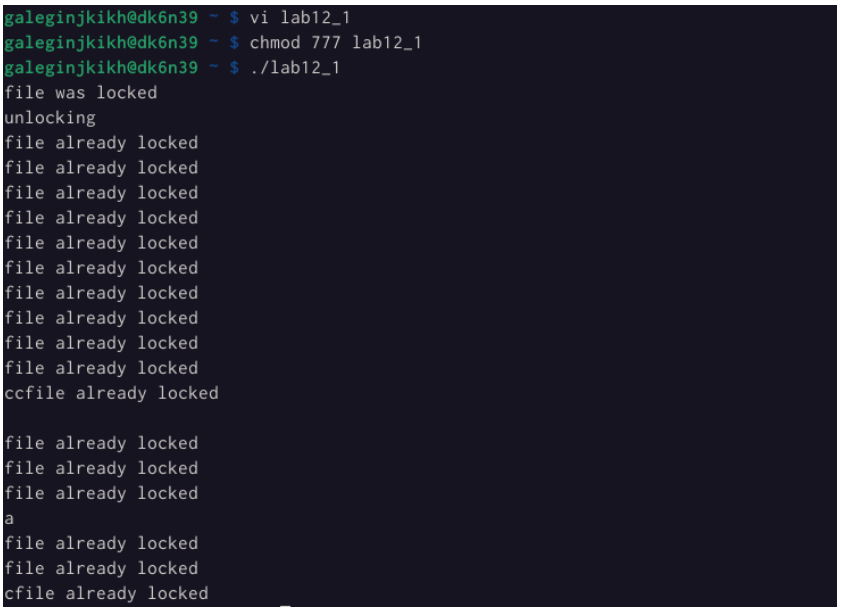

Рис. 3.2: Работа скрипт 1

2. Реализовала команду man с помощью командного файла. Изучила содержимое каталога `/usr/share/man/man1`. В нем находятся архивы текстовых фай-

лов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой less сразу же просмотрев содержимое справки. Командный файл должен получать в виде аргумента командностроки название команды и в виде результата выдавать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет вкаталоге `man1`.(рис. 3.3)(рис. 3.4)

Рис. 3.3: Скрипт 2

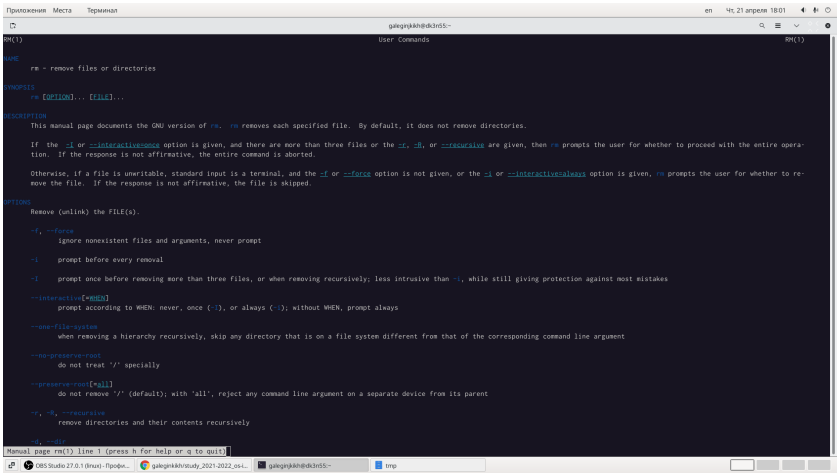

Рис. 3.4: Работа скрипт 2

3. Используя встроенную переменную $RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтя,что $RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.(рис. 3.5)

Рис. 3.5: Скрипт 3

Для наглядности запустила скрипт несколько раз, чтобы увидеть, что генерируются разные случайные пароли.(рис. 3.6)

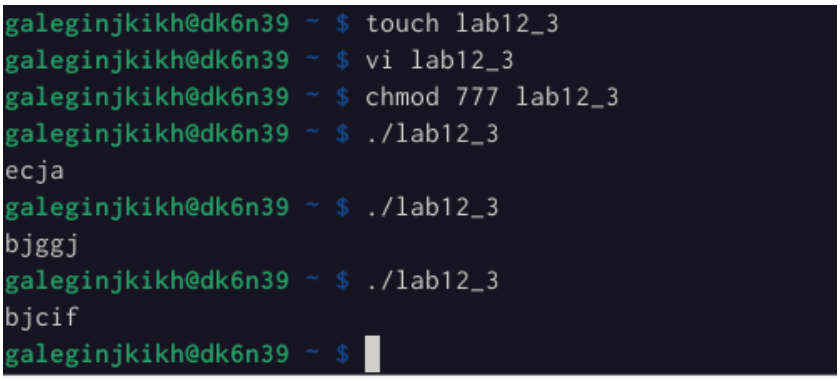

Рис. 3.6: Работа скрипт 3

4 Вывод

В ходе лабораторной работы я изучила основы программирования в оболочке ОС UNIX, научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 Контрольные вопросы

1: **Найдите синтаксическую ошибку в следующей строке: while [$1 != “exit”]**

$1. Так же между скобками должны быть пробелы. В противном случае скобки и рядом стоящие символы будут восприниматься как одно целое

2: **Как объединить (конкатенация) несколько строк в одну?**

3: **Найдите информацию об утилите seq. Какими иными способами можно реализовать её функционал при программировании на bash?**

seq - выдает последовательность чисел. Реализовать ее функционал можно командой for n in {1..5} do done

4: **Какой результат даст вычисление выражения $((10/3))?**

3

5: **Укажите кратко основные отличия командной оболочки zsh от bash.**

Zsh очень сильно упрощает работу. Но существуют различия. Например, в zsh после for обязательно вставлять пробел, нумерация массивов в zsh начинается с 1 (что не особо удобно на самом деле).

Если вы собираетесь писать скрипт, который легко будет запускать множество разработчиков, то я рекомендуется Bash. Если скрипты вам не нужны - Zsh (более простая работа с файлами, например)

6: **Проверьте, верен ли синтаксис данной конструкции for ((a=1; a <= LIMIT; a++))**

Верен

7: **Сравните язык bash с какими-либо языками программирования. Какие**

преимущества у bash по сравнению с ними? Какие недостатки?

Bash позволяет очень легко работать с файловой системой без лишних конструкций (в отличи от обычного языка программирования). Но относительно обычных языков программирования bash очень сжат. Тот же Си имеет гораздо более широкие возможности для разработчика.