

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 13

дисциплина: операционные системы

Студент: Соболевский Денис Андреевич

Группа: НФИбд-02-20

МОСКВА

2021 г.

Цель работы:

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

Теоретическое введение:

В данной лабораторной работе нам предстоит научиться писать командные файлы и использовать их на практике. Для этого нам необходимо ознакомиться с некоторой теорией.

Командные процессоры (оболочки)

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 Сподобный 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/Linuxподобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже.

Переменные в языке программирования bash

Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда

```
mark=/usr/andy/bin
```

присваивает значение строки символов /usr/andy/bin переменной mark типа строка символов.

Использование:

```
mv afile ${mark}
```

переместит файл afile из текущего каталога в каталог с абсолютным полным именем /usr/andy/bin.

Использование значения, присвоенного некоторой переменной, называется подстановкой.

Команды read и echo

Команда read позволяет записать значение для переменной с клавиатуры. Она имеет следующий синтаксис:

```
read
```

Команда echo выводит текст на экран, если имеет вид:

```
echo "Some text"
```

В данном случае она выведет на экран Some text.

С помощью данной команды также можно вывести на экран содержимое, например, переменных:

echo

С прочей теорией и основами языка bash можно ознакомиться в материалах к лабораторной работе №11[1].

Также в ходе выполнения заданий лабораторной работы я столкнулся в необходимости изучения дополнительных материалов, а именно:

- циклы if[2]
- массивы[3]
- утилита test[4]

Выполнение работы:

Задание 1

Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени t2<t1, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустить 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.

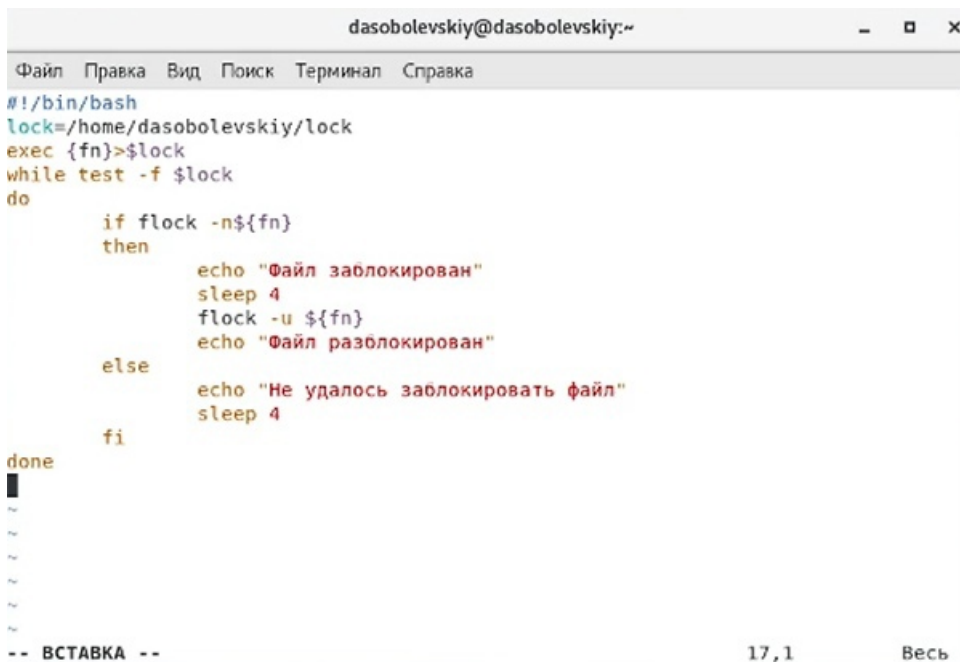
Создадим командный файл lock.sh, который будет реализовывать упрощенный механизм семафоров, с помощью команды vi lock.sh, он сразу же откроется, начнем его написание (рисунок 1).

Задаем переменную lock, в которой хранится полный путь к файлу, который мы будем блокировать (даже если такого файла не существует, он будет создан по средствам последующих команд, выполняющихся в командном файле).

Далее присваиваем через команду exec {fn}>\$lock дескриптор, для возможности работы с командой flock в дальнейшем.

Входим в бесконечный цикл while, условием которого является то, что lock является обычным файлом - test -f. В данном цикле имеется условие if, если файл уже заблокирован - flock -n \${fn} (где \${fn} дескриптор, номер нашего файла), - выводим сообщение об этом и выжидаем 4 секунды, имитируя внутреннюю работу с файлом. После этого разблокируем файл и выводим сообщение об этом. Если же файл заблокировать не получается, тоже выводим об этом сообщение.

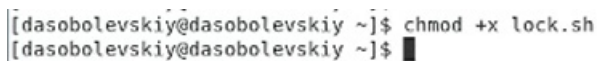
Рисунок 1:



```
dasobolevskiy@dasobolevskiy:~  
Файл Правка Вид Поиск Терминал Справка  
#!/bin/bash  
lock=/home/dasobolevskiy/lock  
exec {fn}>$lock  
while test -f $lock  
do  
    if flock -n${fn}  
    then  
        echo "Файл заблокирован"  
        sleep 4  
        flock -u ${fn}  
        echo "Файл разблокирован"  
    else  
        echo "Не удалось заблокировать файл"  
        sleep 4  
    fi  
done  
-- ВСТАВКА -- 17,1 Весь
```

Теперь проверим правильность работы нашего командного файла (рисунок 2). Для этого сначала добавим права на выполнение chmod +x lock.sh.

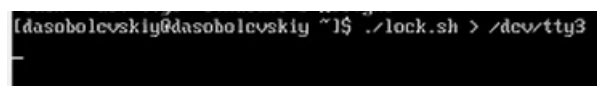
Рисунок 2:



```
[dasobolevskiy@dasobolevskiy ~]$ chmod +x lock.sh  
[dasobolevskiy@dasobolevskiy ~]$
```

Теперь откроем текстовую консоль tty2 нажатием клавиш Ctrl + Alt + F2. В ней вызовем наш командный файл и переадресуем вывод в третью текстовую консоль ./lock.sh > /dev/tty3 (рисунок 3).

Рисунок 3:



```
[dasobolevskiy@dasobolevskiy ~]$ ./lock.sh > /dev/tty3
```

Открываем консоль tty3 и видим, что в ней действительно выполняется командный файл - файл блокируется и разблокируется (рисунок 4).

Рисунок 4:

```
CentOS Linux 7 (Core)
Kernel 3.10.0-1160.el7.x86_64 on an x86_64

dasobolevskiy login: dasobolevskiy
Password:
Last login: Sat Jun  5 12:28:52 on tty2
[dasobolevskiy@dasobolevskiy ~]$ файл заблокирован
файл разблокирован
файл заблокирован
файл разблокирован
файл заблокирован
файл разблокирован
файл заблокирован
файл разблокирован
файл заблокирован
файл разблокирован
файл заблокирован
файл разблокирован
файл заблокирован
файл разблокирован
файл заблокирован
файл разблокирован
файл заблокирован
файл разблокирован
```

Запустим наш файл в консоли tty4 в фоновом режиме ./lock.sh &(рисунок 5).

Рисунок 5:

```
CentOS Linux 7 (Core)
Kernel 3.10.0-1160.el7.x86_64 on an x86_64

dasobolevskiy login: dasobolevskiy
Password:
Last login: Sat Jun  5 12:36:54 on tty3
[dasobolevskiy@dasobolevskiy ~]$ ./lock.sh &
[1] 3645
[dasobolevskiy@dasobolevskiy ~]$ Не удалось заблокировать файл
Не удалось заблокировать файл
Не удалось заблокировать файл
Не удалось заблокировать файл
Не удалось заблокировать файл
Не удалось заблокировать файл
Не удалось заблокировать файл
Не удалось заблокировать файл
```

Таким образом мы наблюдаем, как запущенный файл в привилегированном режиме блокирует файл, работает с ним и разблокирует, а запущенный в фоновом режиме элементарно не успевает что-либо сделать с файлом, поскольку время его запланированной блокировки совпадает со временем, когда привилегированный запуск работает с файлом.

Задание 2

Реализовать команду man с помощью командного файла. Изучите содержимое каталога /usr/share/man/man1. В нем находятся архивы текстовых файлов, 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 командной строки название команды и в виде результата выдавать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1.

Создадим командный файл man13.sh, который будет релаизовывать команду man, с помощью команды vi man13.sh, он сразу же откроется, начнем его написание (рисунок 6).

2 строка - переходим в каталог /usr/share/man/man1. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд, которые мы будем просматривать. Зададим переменную command - имя команды, которое мы будем вводить с клавиатуры. Далее на экран командой echo выведем сообщение о том, что нам необходимо ввести имя команды, информацию о которой мы хотим узнать. Вводим ее с клавиатуры через read и снова выводим сообщение о том, что далее будет показана информация по данной команде (однако это ненужно, поскольку информация будет показана не в терминале, а в отдельном окне как привывове man). Командой less просмотрим содержимое справки, используя указатель на имя команды \$.

Рисунок 6:

```
dasobolevskiy@dasobolevskiy:~  
Файл Правка Вид Поиск Терминал Справка  
#!/bin/bash  
cd /usr/share/man/man1  
command=""  
echo "Enter command:"  
read command  
echo "Command information:"  
less $command*
```

Теперь проверим правильность работы нашего командного файла (рисунок 7). Для этого сначала добавим права на выполнение `chmod +x man13.sh`. Далее вызовем наш файл для проверки в качестве команды `./man13.sh`. Видим, что он выводит все сообщения так, как было задумано, после ввода команды (мы будем просматривать информацию о команде `cp`), действительно выводит справку о ней (рисунок 8). Нажимаем клавишу `q`, чтобы закончить просмотр справки.

Рисунок 7:

```
[dasobolevskiy@dasobolevskiy ~]$ chmod +x man13.sh  
[dasobolevskiy@dasobolevskiy ~]$ ./man13.sh  
Enter command:  
cp
```

Рисунок 8:

```
dasobolevskiy@dasobolevskiy:~  
Файл Правка Вид Поиск Терминал Справка  
CP(1) User Commands CP(1)  
  
ESC[1mNAMEESC[0m  
cp - copy files and directories  
  
ESC[1mSYNOPSISESC[0m  
ESC[1mcp ESC[22m[ESC[4mOPTIONESC[24m]... [ESC[4m-TESC[24m] ESC[4mSOURCE  
ESC[24m ESC[4mDESTESC[0m  
ESC[1mcp ESC[22m[ESC[4mOPTIONESC[24m]... ESC[4mSOURCEESC[24m... ESC[4mDIR  
ECTORYESC[0m  
ESC[1mcp ESC[22m[ESC[4mOPTIONESC[24m]... ESC[4m-tESC[24m ESC[4mDIRECTORY  
ESC[24m ESC[4mSOURCEESC[24m...  
  
ESC[1mDESCRIPTIONESC[0m  
Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.  
  
Mandatory arguments to long options are mandatory for short options  
too.  
  
ESC[1m-aESC[22m, ESC[1m--archiveESC[0m  
same as ESC[1m-dR --preserveESC[22mESC[4mallESC[0m  
cp.1.gz (file 1 of 11)
```

Задание 3

Используя встроенную переменную `$RANDOM`, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что `$RANDOM` выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

Создадим командный файл `random_line.sh`, который будет релаизовывать генерацию строк с randomным надором букв латинского алфавита, с помощью команды `vi random_line.sh`, он сразу же откроется, начнем его написание (рисунок 9).

Рисунок 9:

```
dasobolevskiy@dasobolevskiy:~  
Файл Правка Вид Поиск Терминал Справка  
#!/bin/bash  
  
echo $RANDOM | tr '0-9' '[a-zA-Z]'
```

Проверим работу нашего командного файла (рисунок 10). Для этого сначала добавим права на выполнение `chmod +x random_line.sh`. Далее вызовем наш файл для проверки в качестве команды `./random_line.sh`. Повторим это несколько раз, видим, что на выход получаем строки разной длины с произвольным набором букв - задача выполнена.

Рисунок 10:

```
[dasobolevskiy@dasobolevskiy ~]$ chmod +x random_line.sh
[dasobolevskiy@dasobolevskiy ~]$ ./random_line.sh
ciie
[dasobolevskiy@dasobolevskiy ~]$ ./random_line.sh
beabc
[dasobolevskiy@dasobolevskiy ~]$
```

Вывод:

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

Библиография:

- [1] [Лабораторная работа №11](#)
- [2] [Циклы if](#)
- [3] [Использование массивов в bash](#)
- [4] [Утилита test](#)