Лабораторная работа 12

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Генералов Даниил, НПИ-01-21, 1032212280

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Целью данной работы является:

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

2 Задание

Требуется написать 3 командных файла:

- берет эксклюзивное управление файлом и держит его некоторое время
- реализует отображение `man`-страниц
- выводит последовательность случайных символов

3 Теоретическое введение

Командный процессор (*shell*) – это программа на Unix-системах, которая принимает ввод от пользователя и исполняет инструкции. Помимо интерактивного использования, она может исполнять список команд, заданный в файле, и она обладает набором команд, достаточным для написания программ разной степени сложности. В этой работе мы продолжаем рассматриваьб этот функционал командного процессора, составляя несколько командных файлов, выполняющих определенные действия.

4 Выполнение лабораторной работы

Первая программа представлена на рис. 4.1. Она принимает четыре аргумента: имя файла, время ожидания для получения блокировки и время, которое следует держать блокировку. После получения соответствующих аргументов, программа пытается получить блокировку на файл с помощью команды `flock`. Если это удается сделать за указанное время, программа ждет заданное время и затем освобождает блокировку. Если нет, то программа выводит сообщение об ошибке и завершается. Оба случая показаны на рис. 4.2.

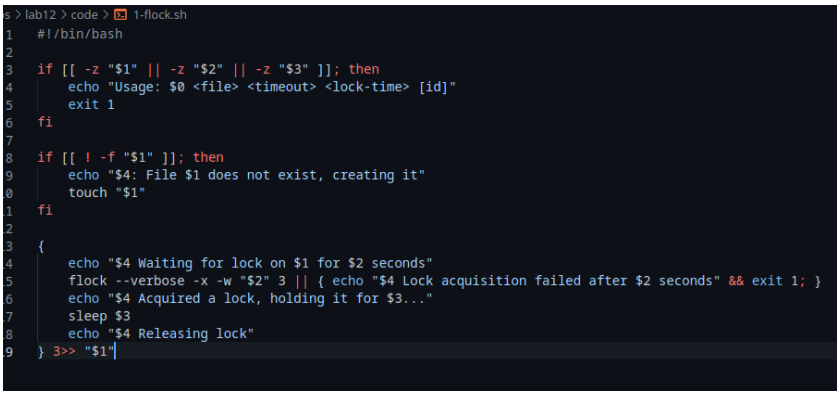


Рис. 4.1: Программа блокировки файлов

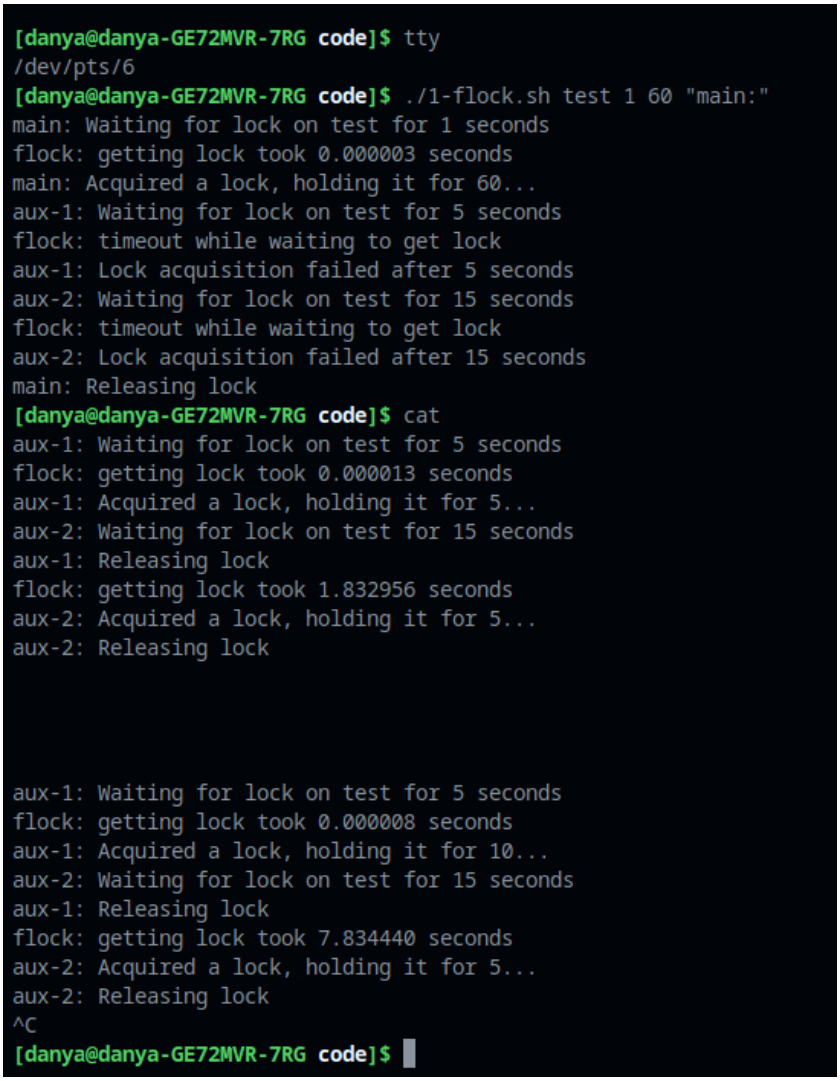


Рис. 4.2: Результат работы программы блокировки файлов

Вторая программа представлена на рис. 4.3. Она принимает один аргумент: имя `man`-страницы. Если такая `man`-страница существует в стандартном расположении в разделе 1, то эта страница читается с помощью `zcat`, разархивируя ее, а затем обрабатывается через `groff` – современную реализацию `roff`-инфраструктуры – и показывается через `less`. Чтобы показать это на рис. 4.4, необходимо передать вывод `less` через какую-то программу, которая не требует эксклюзивного доступа к терминалу – я использую `head`.

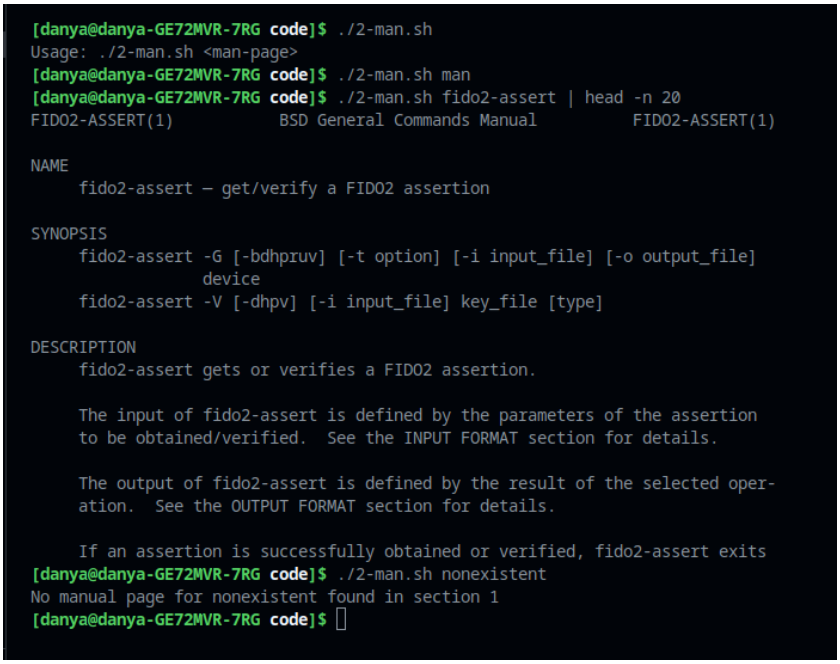
```
#!/bin/bash

if [[ -z "$1" ]]; then
    echo "Usage: $0 <man-page>"
    exit 1
fi

if [[ ! -f "/usr/share/man/man1/$1.1.gz" ]]; then
    echo "No manual page for $1 found in section 1"
    exit 1
fi

zcat "/usr/share/man/man1/$1.1.gz" | groff -T utf8 -man | less -R
```

Рис. 4.3: Программа показа страницы



```
[danya@danya-GE72MVR-7RG code]$ ./2-man.sh
Usage: ./2-man.sh <man-page>
[danya@danya-GE72MVR-7RG code]$ ./2-man.sh man
[danya@danya-GE72MVR-7RG code]$ ./2-man.sh fido2-assert | head -n 20
FIDO2-ASSERT(1)           BSD General Commands Manual          FIDO2-ASSERT(1)

NAME
     fido2-assert – get/verify a FIDO2 assertion

SYNOPSIS
     fido2-assert -G [-bdhpruv] [-t option] [-i input_file] [-o output_file]
                  device
     fido2-assert -V [-dhpv] [-i input_file] key_file [type]

DESCRIPTION
     fido2-assert gets or verifies a FIDO2 assertion.

     The input of fido2-assert is defined by the parameters of the assertion
     to be obtained/verified.  See the INPUT FORMAT section for details.

     The output of fido2-assert is defined by the result of the selected oper-
     ation.  See the OUTPUT FORMAT section for details.

     If an assertion is successfully obtained or verified, fido2-assert exits
[danya@danya-GE72MVR-7RG code]$ ./2-man.sh nonexistent
No manual page for nonexistent found in section 1
[danya@danya-GE72MVR-7RG code]$
```

Рис. 4.4: Результат работы программы показа страниц

Третья программа представлена на рис. 4.5. Она не принимает аргументов и выполняется бесконечно. Она получает значение переменной `$RANDOM`, которое равномерно распределено между значениями 0 и 32767, затем вычисляет остаток от деления этого числа на 26, а затем выводит символ ASCII, соответствующий маленькой букве с таким номером. Пример вывода программы на рис. 4.6 разделен символами новой строки, чтобы он был виден на одном экране.

labs > lab12 > code > 3-random.sh

```
#!/bin/bash

while true; do
    charind=$((RANDOM % 26 + 97))
    charcode=`printf '%x' $charind`
    char=`printf "\x$charcode"`
    echo -n $char
    sleep 0.1
done
```

Рис. 4.5: Программа вывода символов

```
[danya@danya-GE72MVR-7RG code]$ ./3-random.sh
xboncgnuqflpdhkklbqxuzjujzgrwfcizcsootbycjqqzzhvaarl
vdwhltqgyhetwqd
zjgxc
x
wgshdoutedhhqsiflylggfvhksxtlxqtwqxdlgeug
nniuexil
pejrlcblbrouluiswjrztlhxkpjdmgxugxckfopxjsnrqg
hl
iusz
qpgf
nhrlbpaditvbkbetarfpa^C
[danya@danya-GE72MVR-7RG code]$
```

Рис. 4.6: Результат работы программы вывода символов

5 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке:

```
while [$1 != "exit"]
```

Здесь предполагается использовать проверку условия, но условие должно быть написано внутри двойных квадратных скобок (`[[ $1 != "exit" ]]`).

2. Как объединить (конкатенация) несколько строк в одну?

Для этого есть несколько способов:

```
A = "Hello, "
B = "World!"
C1 = "$A$B"
C2 = $(echo -n $A $B)
C3 = `printf "%s%s" "$A" "$B"`
```

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

Команда `seq` предоставляет возможность получить последовательность чисел от начального до конечного значения включительно:

```
$ seq 1 10 2
1
3
```

```
5
7
9
```

Если все эти параметры являются константами, можно использовать синтаксис списков:

```
$ echo {1..10..2}
1 3 5 7 9
```

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

Это выражение имеет значение `3`. Это потому, что оператор `/` в арифметике в Bash округляет ответ до целого числа.

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

Оболочка `zsh` имеет несколько отличающийся интерфейс от `bash`, и эти отличия могут делать ее более удобной для интерактивного использования. Например, используя команду `z`, можно перейти к часто используемой директории по имени. Помимо этого, `zsh` имеет большую поддержку для расширения функционала, например для создания плагинов для отображения текущего состояния `git`-репозитория.

С точки зрения API программирования, `zsh` поддерживает дробную арифметику, и использует несколько другой механизм от `bash` для определения опций автодополнения.

6. Проверьте, верен ли синтаксис данной конструкции

```
for ((a=1; a <= LIMIT; a++))
```

Да, такой синтаксис работает в `bash`.

7. Сравните язык bash с какими-либо языками программирования. Какие преимущества у bash по сравнению с ними? Какие недостатки?

Одной из отличительных особенностей `sh`-подобных языков является то, что они имеют только один тип данных – строки. Когда в языках вроде C, Python, и даже JavaScript и PHP есть различные типы данных, имеющих разные правила работы, в `bash` все переменные – в том числе числа и списки – это строки. Является ли это преимуществом или недостатком – спорный вопрос.

Среди преимуществ можно считать то, что этот язык доступен везде – любая POSIX-совместимая система будет иметь возможность использовать его как основной язык программирования, даже когда другие компиляторы и интерпретаторы отсутствуют.

Среди недостатков можно считать то, что этот язык поддерживает только одну парадигму программирования – императивно-процедурное. Когда языки вроде C/C++ в основном императивные, но имеют элементы объектно-ориентированного программирования, и Python по умолчанию объектно-ориентированный, но может иметь элементы функционального программирования, а Haskell по умолчанию функциональный, но может использоваться как императивный, в Bash же есть только одна парадигма, от которой нельзя отойти.

6 Выводы

В этой лабораторной работе мы закончили знакомство с основами программирования в Bash. Мы рассмотрели продвинутые концепции программирования, которые позволяют нам писать более сложные программы в Bash. В дальнейщем этот навык может оказаться весьма полезным.