

Отчет по лабораторной работа №12

**Программирование в командном процессоре ОС UNIX. Расширенное
программирование**

Лушин Артем Андреевич

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Контрольные вопросы	10
4	Выводы	12
	Список литературы	13

Список иллюстраций

2.1	Текст 1 скрипта	6
2.2	Результат работы	7
2.3	Текст 2 скрипта	7
2.4	Запуск программы	7
2.5	Результат работы	8
2.6	Текст 3 скрипта	8
2.7	Результат работы	9

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 Выполнение лабораторной работы

- 1) Я написал командный файл, реализующий упрощенный механизм семафоров. Командный файл должен в течение 3 секунд дожидаться ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени, также выдавая информация о том, что ресурс используется соответствующим командным файлом.

```
#!/bin/bash
LOCKFILE="./lock.file"
exec {fn}>$LOCKFILE

if test -f "$LOCKFILE"
then
....
    while.
<----->[ 1 = 1
    do
<----->if flock -n ${fn}
<----->then
<----->    echo "Файл заблокирован"
<----->    sleep 3
<----->    echo "Файл разблокирован"
<----->    flock -u ${fn}
<----->else
<----->    echo "Файл заблокирован"
<----->    sleep 3
<----->fi
        done
fi
```

Рис. 2.1: Текст 1 скрипта

```
[aalushin@fedora 12]$ ./1
Файл заблокирован
Файл разблокирован
Файл заблокирован
```

Рис. 2.2: Результат работы

- 2) Я реализовал команду `man` с помощью командного файла. Изучил содержимое каталога `/usr/share/man/man1`. Командный файл получает название команды и выводит справку о ней, если же такой команды нет, то он выводит сообщение об отсутствии такой команды.

```
2 [----] 40 L: [ 1+11 12/ 15] *(199 / 233)
#!/bin/bash.
command=""

while getopts :n: opt
do
    case $opt in
        n) command="$OPTARG";;
        esac
    done

    if test -f "/usr/share/man/man1/$command.1.gz"
    then less /usr/share/man/man1/$command.1.gz
    else
        echo "no such command"
    fi
fi
```

Рис. 2.3: Текст 2 скрипта

```
[aalushin@fedora 12]$ ./2 -n kill
[11]+  Остановлен ./2 -n kill
[aalushin@fedora 12]$
```

Рис. 2.4: Запуск программы

```
KILL(1)                                User Commands                                KILL(1)

ESC[1mNAMEESC[0m
kill - terminate a process

ESC[1mSYNOPSISESC[0m
ESC[1mkill ESC[22mESC[1mESC[4mESC[22msignalESC[24mESC[1m-s ESC[4mESC[22msignalESC[24mESC[1m-pESC[22m] (ES
ESC[1m-q ESC[4mESC[22mvalueESC[24m] ESC[1m-pESC[22m] ESC[1m--timeout ESC[4mESC[22millisecondESC[24mESC[4msignalESC
ESC[24m] ESC[1m--ESC[22m] ESC[4mpidESC[24m]ESC[4mnameESC[24m...

ESC[1mkill -l ESC[22mESC[4mnumberESC[24m] | ESC[1m-ESC[0m

ESC[1mDESCRIPTIONESC[0m
The command ESC[1mkill ESC[22msends the specified ESC[4msignalESC[24m to the specified processes or process gro
ups.

If no signal is specified, the ESC[1mTERMESC[22msignal is sent. The default action for this signal is to termi
nate the
process. This signal should be used in preference to the ESC[1mKILL ESC[22msignal (number 9), since a process m
ay install
a handler for the TERM signal in order to perform clean-up steps before terminating in an orderly fashion.
If a process does not terminate after a ESC[1mTERMESC[22msignal has been sent, then the ESC[1mKILL ESC[22msign
al may be used; be
aware that the latter signal cannot be caught, and so does not give the target process the opportunity to
perform any clean-up before terminating.

Most modern shells have a builtin ESC[1mkill ESC[22mcommand, with a usage rather similar to that of the command
described
here. The ESC[1m--allESC[22m, ESC[1m--pidESC[22m, and ESC[1m--queue ESC[22moptions, and the possibility to spec
ify processes by command name, are
local extensions.

If ESC[4msignalESC[24m is 0, then no actual signal is sent, but error checking is still performed.

ESC[1mARGUMENTSESC[0m
The list of processes to be signaled can be a mixture of names and PIDs.
```

Рис. 2.5: Результат работы

- 3) Используя встроенную переменную, написал командный файл, который генерирует случайную последовательность букв латинского алфавита и чисел.

```
4 [-----] 62 L: [ 1+ 1 2/ 2] *(74 / 740) <EOF>
#!/bin/bash
cat /dev/urandom | tr -dc "a-zA-Z0-9" | fold -w 10 | head -n 1
```

Рис. 2.6: Текст 3 скрипта


```
[aalushin@fedora 12]$ ./4
yLCOSTOc1R
[aalushin@fedora 12]$ ./4
FNhQDc3rnL
[aalushin@fedora 12]$ ./4
JzQhdz0lVg
[aalushin@fedora 12]$ ./4
GQ6qgXSEKv
[aalushin@fedora 12]$ ./4
rEE3TMekOo
[aalushin@fedora 12]$ ./4
50m9PP7f2n
[aalushin@fedora 12]$ ./4
6iI53sLWWO
[aalushin@fedora 12]$ ./4
UIyn9qEV8p
[aalushin@fedora 12]$ ./4
kXxJmKwcih
[aalushin@fedora 12]$
```

Рис. 2.7: Результат работы

3 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?

```
[aalushin@fedora ~]$ cat file.txt | xargs | sed -e 's/\./.\n/g'
```

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 рекомендую `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 отличие от обычного языка программирования). Но относительно обычных языков программирования `bash` очень сжат. Тот же `C` имеет гораздо более широкие возможности для разработчика.

4 Выводы

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

Список литературы