

Отчёт к лабораторной работе №13

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

Кекишева Анастасия Дмитриевна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
3.1	Выполнение 1-го пункта задания	8
3.2	Выполнение 2-го пункта задания	10
3.3	Выполнение 3-го пункта задания	12
4	Вывод	14
5	Библиография	15
6	Контрольные вопросы	16

Список таблиц

Список иллюстраций

3.1	Командный файл, реализующий упрощённый механизм семафоров	8
3.2	Результат выполнения командного файла	9
3.3	Командный файл реализующий команду <code>map</code>	10
3.4	Содержимое каталога <code>/usr/share/man/man1</code>	10
3.5	Вызов командного файла программы	11
3.6	Результат работы программы: справка команды <code>mv</code>	11
3.7	Командный файл, генерирующий случайную последовательность букв латинского алфавита	12
3.8	Результат работы программы	13

1 Цель работы

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

2 Задание

Выполнить данные пункты:

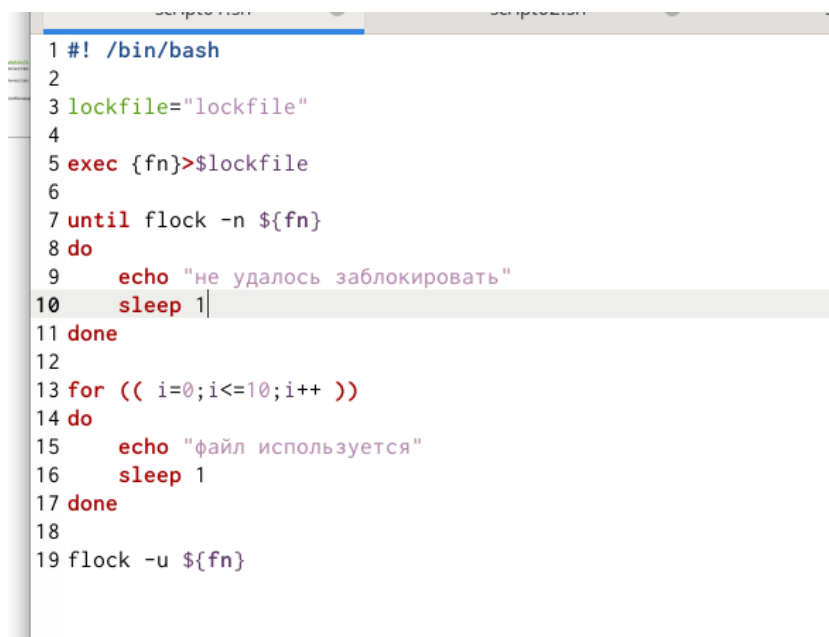
1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию 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. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.
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`.
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.

3 Выполнение лабораторной работы

Перед выполнением лабораторной работы я хорошо ознакомилась с теоретическим материалом для её выполнения Ссылка 1

3.1 Выполнение 1-го пункта задания



```
1 #!/bin/bash
2
3 lockfile="lockfile"
4
5 exec {fn}>$lockfile
6
7 until flock -n ${fn}
8 do
9     echo "не удалось заблокировать"
10    sleep 1
11 done
12
13 for (( i=0;i<=10;i++ ))
14 do
15     echo "файл используется"
16     sleep 1
17 done
18
19 flock -u ${fn}
```

Рис. 3.1: Командный файл, реализующий упрощённый механизм семафоров

Командный файл в течение некоторого времени t_1 дожидается освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использует файл в течение некоторого времени $t_2 < t_1$, также выдавая информацию о том,

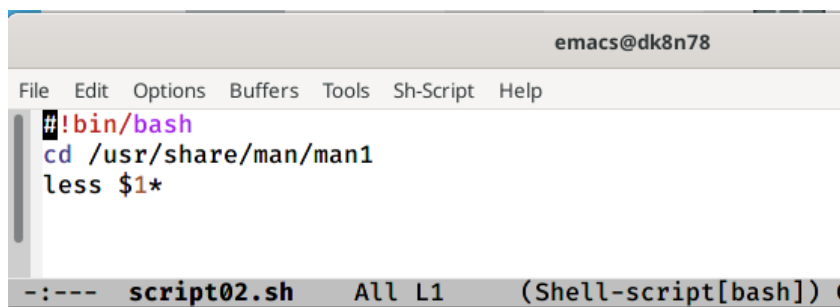
что ресурс используется соответствующим командным файлом (процессом). Цикл `until` крутится до тех пор, пока условие не станет истинным (рис. 3.1).

```
для более детальной информации смотрите файл /usr/share/doc/iptables/README.Debian
adkekisheva@dk8n78 ~ $ ./script01.sh
файл используется
файл используется
файл используется
файл используется
файл используется
файл используется
файл используется
файл используется
файл используется
файл используется
файл используется
adkekisheva@dk8n78 ~ $ ./script01.sh
не удалось заблокировать
не удалось заблокировать
не удалось заблокировать
не удалось заблокировать
не удалось заблокировать
не удалось заблокировать
не удалось заблокировать
не удалось заблокировать
не удалось заблокировать
не удалось заблокировать
не удалось заблокировать
^C
adkekisheva@dk8n78 ~ $
```

Рис. 3.2: Результат выполнения командного файла

Запустила файл в двух консолях и проверила правильность работы. (рис. 3.2). К сожалению, нам не удалось её исправить. Скорее всего, это ошибка в системе, файл не выполняет свои функции.

3.2 Выполнение 2-го пункта задания



```
emacs@dk8n78
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
cd /usr/share/man/man1
less $1*
-:--- script02.sh All L1 (Shell-script[bash])
```

Рис. 3.3: Командный файл реализующий команду man



```
adkeisheva@dk8n78 /usr $ cd share/man/man1
adkeisheva@dk8n78 /usr/share/man/man1 $ ls
411toppm.1.bz2          nvme-lnvm-diag-bbtl.1.bz2
7z.1.bz2               nvme-lnvm-diag-set-bbtl.1.bz2
7za.1.bz2              nvme-lnvm-factory.1.bz2
7zr.1.bz2              nvme-lnvm-id-ns.1.bz2
a2ps.1.bz2             nvme-lnvm-info.1.bz2
a2x.1.bz2              nvme-lnvm-init.1.bz2
a52dec.1.bz2           nvme-lnvm-list.1.bz2
aacplusenc.1.bz2       nvme-lnvm-remove.1.bz2
ab.1.bz2               nvme-netapp-ontapdevices.1.bz2
aclocal-1.11.1.bz2     nvme-netapp-smdevices.1.bz2
aclocal-1.12.1.bz2     nvme-ns-descs.1.bz2
aclocal-1.13.1.bz2     nvme-ns-rescan.1.bz2
aclocal-1.14.1.bz2     nvme-read.1.bz2
aclocal-1.15.1.bz2     nvme-reset.1.bz2
aclocal-1.16.1.bz2     nvme-resv-acquire.1.bz2
acconnect.1.bz2        nvme-resv-register.1.bz2
acyclic.1.bz2          nvme-resv-release.1.bz2
adddebug.1.bz2         nvme-resv-report.1.bz2
addedge.1.bz2          nvme-sanitize.1.bz2
addftinfo.1.bz2        nvme-sanitize-log.1.bz2
addrinfo.1.bz2         nvme-security-recv.1.bz2
advdef.1.bz2           nvme-security-send.1.bz2
advmng.1.bz2           nvme-self-test-log.1.bz2
```

Рис. 3.4: Содержимое каталога /usr/share/man/man1

Реализовала команду man с помощью командного файла (рис. 3.3). Изучила содержимое каталога /usr/share/man/man1 (рис. 3.4), 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 получает в виде аргумента командной строки название команды и в результате выдаёт справку об этой команде или сообщение об отсутствии каталога с информацией об этой команде, если команда набрана не правильно.

```
adkekisheva@dk8n78 ~ $ bash script02.sh
adkekisheva@dk8n78 ~ $ bash script02.sh mv
adkekisheva@dk8n78 ~ $ bash script02.sh find
adkekisheva@dk8n78 ~ $ bash script02.sh findpo
findpo*: Нет такого файла или каталога
adkekisheva@dk8n78 ~ $
```

Рис. 3.5: Вызов командного файла программы

```
MV(1)                                User Commands                                MV(1)
NAME
    mv - move (rename) files
SYNOPSIS
    mv [OPTION]... [-T] SOURCE DEST
    mv [OPTION]... SOURCE... DIRECTORY
    mv [OPTION]... -t DIRECTORY SOURCE...
DESCRIPTION
    Rename SOURCE to DEST, or move SOURCE(s) to DIRECTORY.

    Mandatory arguments to long options are mandatory for short options
    too.

    --backup[=CONTROL]
        make a backup of each existing destination file
    -b
        like --backup but does not accept an argument
    -f, --force
        do not prompt before overwriting
```

Рис. 3.6: Результат работы программы: справка команды mv

После, проверила выполнение командного файла (рис. 3.5), вызвав его командой `bash`. И получила информацию о команде, которую ввела в командной строке (рис. 3.6). Если нет команды, то вылезет сообщение об ошибке.

3.3 Выполнение 3-го пункта задания

```
1 #!/bin/bash
2 echo "Введите количество комбинаций: "
3 read number
4 M=number
5 echo "Введите количество букв для генерирования комбинаций"
6 read num
7 c=1
8 d=1
9 echo
10 echo "Рандомные комбинации букв:"
11 while (($c!=($M+1)))
12 do
13     echo $d
14     echo $(for((i=1;i<=$num;i++));
15 do printf '%s' "${RANDOM:0:1}"; done) | tr '0-9' '[a-z]'
16     ((c+=1))
17     ((d+=1))
18 done
19
```

Рис. 3.7: Командный файл, генерирующий случайную последовательность букв латинского алфавита

Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита (рис. 3.7). Учла, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767. Для начала я прошу пользователя ввести количество комбинаций для вывода и количество букв для генерирования этих комбинаций. Далее в цикле while я регулирую количество комбинаций и пока $c \neq M+1$ я делаю: цикл for, который будет регулировать количество букв и осуществляю печать рандомной комбинации, используя встроенную переменную \$RANDOM, а также применяю конвеер с командой tr [Ссылка 2](#), которая обрабатывает текст посимвольно и задаю для обработки цифры от 0 до 9 и английский алфавит.

```

adkekisheva@dk8n78 ~ $ bash script03.sh
Введите количество комбинаций:
4
Введите количество букв для генерирования комбинаций
6

Рандомные комбинации букв:
1
bfbbbg
2
bchbid
3
cbgcid
4
bbbbdc
adkekisheva@dk8n78 ~ $ bash script03.sh
Введите количество комбинаций:
3
Введите количество букв для генерирования комбинаций
2

Рандомные комбинации букв:
1
fb
2
cb
3
cg

```

Рис. 3.8: Результат работы программы

Запустила программу, ввела необходимые данные и получила введённое количество рандомных сочетаний букв. (рис. 3.8)

4 Вывод

Я продолжила изучение основ программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 Библиография

1. Ссылка 1
2. Ссылка 2

6 Контрольные вопросы

1. В строке `while [$1 != "exit"]` квадратные скобки надо заменить на двойные круглые: `while (($1 != "exit"))`.

2. Есть несколько видов конкатенации строк. Например:

- `VAR1="Hello,"`
- `VAR2=" World"`
- `VAR3="VAR1VAR2"`
- `echo "$VAR3"`

3. Команда `seq` выводит последовательность целых или действительных чисел, подходящую для передачи в другие программы. В `bash` можно использовать `seq` с циклом `for`, используя подстановку команд. Например:

```
$ for i in $(seq 1 0.5 4)
```

```
do
```

```
echo "The number is $i"
```

```
done
```

4. Результатом вычисления выражения `$((10/3))` будет число 3.

5. Список того, что можно получить, используя `Z Shell` вместо `Bash`:

Встроенная команда `zmv` поможет массово переименовать файлы/директории, например, чтобы добавить `.txt` к имени каждого файла, запустите `zmv -C '(*)(#q.)' '$1.txt'`.

Утилита `zcalc` — это замечательный калькулятор командной строки, удобный способ считать быстро, не покидая терминал.

Команда `zparseopts` — это однострочник, который поможет разобрать сложные варианты, которые предоставляются скрипту.

Команда `autopushd` позволяет делать `popd` после того, как с помощью `cd`, чтобы вернуться в предыдущую директорию.

Поддержка чисел с плавающей точкой (коей Bash не содержит).

Поддержка для структур данных «хэш».

Есть также ряд особенностей, которые присутствуют только в Bash:

Опция командной строки `-norc`, которая позволяет пользователю иметь дело с инициализацией командной строки, не читая файл `.bashrc`

Использование опции `-rcfile` с `bash` позволяет исполнять команды из определённого файла.

Отличные возможности вызова (набор опций для командной строки)

Может быть вызвана командой `sh`

Bash можно запустить в определённом режиме POSIX. Примените `set -o posix`, чтобы включить режим, или `--posix` при запуске.

Можно управлять видом командной строки в Bash. Настройка переменной `PROMPT_COMMAND` с одним или более специальными символами настроит её за вас.

Bash также можно включить в режиме ограниченной оболочки (с `rbash` или `-restricted`), это означает, что некоторые команды/действия больше не будут доступны:

Настройка и удаление значений служебных переменных `SHELL`, `PATH`, `ENV`, `BASH_ENV`

Перенаправление вывода с использованием операторов `>`, `>|`, `<>`, `>&`, `&>`,

‘»’

Разбор значений SHELLOPTS из окружения оболочки при запуске.

Использование встроенного оператора `exes`, чтобы заменить оболочку другой командой

6. Синтаксис конструкции `for ((a=1; a <= LIMIT; a++))` верен.

7. Язык `bash` и другие языки программирования:

- Скорость работы программ на ассемблере может быть более 50% медленнее, чем программ на `си/си++`, скомпилированных с максимальной оптимизацией;

- Скорость работы виртуальной ява-машины с байт-кодом часто превосходит скорость аппаратуры с кодами, получаемыми трансляторами с языков высокого уровня. Ява-машина уступает по скорости только ассемблеру и лучшим оптимизирующим трансляторам;

- Скорость компиляции и исполнения программ на яваскрипт в популярных браузерах лишь в 2-3 раза уступает лучшим трансляторам и превосходит даже некоторые качественные компиляторы, безусловно намного (более чем в 10 раз) обгоняя большинство трансляторов других языков сценариев и подобных им по скорости исполнения программ;

- Скорость кодов, генерируемых компилятором языка `си` фирмы Intel, оказалась заметно меньшей, чем компилятора GNU и иногда LLVM;

- Скорость ассемблерных кодов `x86-64` может меньше, чем аналогичных кодов `x86`, примерно на 10%;

- Оптимизация кодов лучше работает на процессоре Intel;

- Скорость исполнения на процессоре Intel была почти всегда выше, за исключением языков `лисп`, `эрланг`, `аук` (`gawk`, `mawk`) и `бэш`. Разница в скорости по `бэш` скорее всего вызвана разными настройками окружения на тестируемых системах, а не собственно транслятором или железом. Преимущество Intel особенно заметно на 32-разрядных кодах;

-Стек большинства тестируемых языков, в частности, ява и яваскрипт, поддерживают только очень ограниченное число рекурсивных вызовов. Некоторые трансляторы (gcc, icc, ...) позволяют увеличить размер стека изменением переменных среды исполнения или параметром;

-В рассматриваемых версиях gawk, php, perl, bash реализован динамический стек, позволяющий использовать всю память компьютера. Но perl и, особенно, bash используют стек настолько экстенсивно, что 8-16 ГБ не хватает для расчета `ack(5,2,3)`