

Отчёт по лабораторной работе №14

Операционные системы

Аветисян Алина Эдуардовна

Содержание

1	Цель работы	1
2	Задание	1
3	Теоретическое введение.....	2
4	Выполнение лабораторной работы.....	2
5	Выводы.....	6
6	Ответы на контрольные вопросы.....	6
	Список литературы	7

1 Цель работы

Цель данной лабораторной работы - изучить основы программирования в оболочке ОС UNIX, научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

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 Теоретическое введение

4 Выполнение лабораторной работы

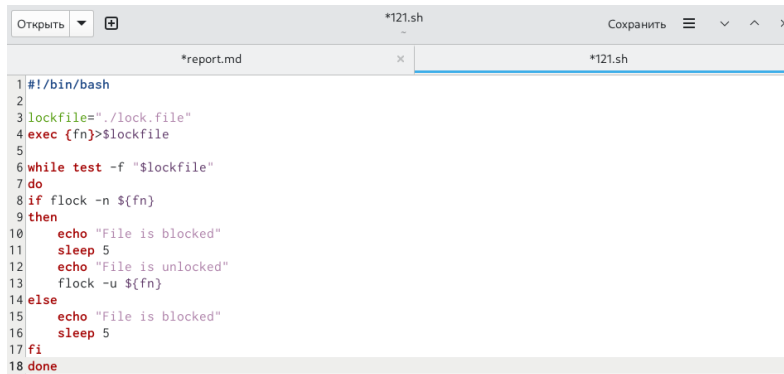
Создаю командный файл для первой программы, пишу ее, проверяю ее работу (рис. fig. 1).

```
aeavetisyan@dk4n65 ~ $ touch 121.sh
aeavetisyan@dk4n65 ~ $ chmod +x 121.sh
aeavetisyan@dk4n65 ~ $ bash 121.sh
```

Рис. 1: Создание и исполнение файла

```
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
```

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. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов (рис. fig. 2).

A screenshot of a code editor window. The title bar shows "Открыть" (Open) on the left and "Сохранить" (Save) on the right, with a file name "*t21.sh" in the center. Below the title bar, there are two tabs: "*report.md" and "*t21.sh". The "*t21.sh" tab is active, showing a shell script. The script starts with a shebang "#!/bin/bash" and defines a lockfile as "./lock.file". It then uses "exec {fn}>\$lockfile" to open a file descriptor. A "while" loop checks if the lockfile exists (-f "\$lockfile"). Inside the loop, an "if" statement checks if the file is locked (flock -n \${fn}). If locked, it echoes "File is blocked" and sleeps for 5 seconds. If not locked, it echoes "File is unlocked" and releases the lock (flock -u \${fn}). The loop ends with "fi" and "done".

```
1 #!/bin/bash
2
3 lockfile="./lock.file"
4 exec {fn}>$lockfile
5
6 while test -f "$lockfile"
7 do
8   if flock -n ${fn}
9   then
10    echo "File is blocked"
11    sleep 5
12    echo "File is unlocked"
13    flock -u ${fn}
14   else
15    echo "File is blocked"
16    sleep 5
17   fi
18 done
```

Рис. 2: Код программы

```
#!/bin/bash
```

```
lockfile="./lock.file"
exec {fn}>$lockfile
```

```
while test -f "$lockfile"
do
if flock -n ${fn}
then
    echo "File is blocked"
    sleep 5
    echo "File is unlocked"
    flock -u ${fn}
else
    echo "File is blocked"
    sleep 5
fi
done
```

Чтобы реализовать команду `man` с помощью командного файла, изучаю 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 (рис. fig. 3).

```
msgfmt.1.bz2      ziptool.1.bz2
msggrep.1.bz2     zless.1.bz2
msginit.1.bz2     zlib_decompress.1.bz2
msgmerge.1.bz2    zlib-flate.1.bz2
msgunfmt.1.bz2    zmore.1.bz2
msguniq.1.bz2     znew.1.bz2
msguntypot.1p.bz2 zonetab2pot.py.1.bz2
mshortname.1.bz2  zrun.1.bz2
mshowfat.1.bz2    zsh.1.bz2
msoelim.1.bz2     zshall.1.bz2
ms_print.1.bz2    zshbuiltins.1.bz2
msql2mysql.1.bz2 zshcalsys.1.bz2
ms-sys.1.bz2      zshcompctl.1.bz2
msxlint.1.bz2     zshcompsys.1.bz2
mtools.1.bz2      zshcompwid.1.bz2
mtooltest.1.bz2   zshcontrib.1.bz2
mtrace.1.bz2      zshexpn.1.bz2
mtvtoppm.1.bz2    zshmisc.1.bz2
mtype.1.bz2       zshmodules.1.bz2
mu.1.bz2          zshoptions.1.bz2
mu-add.1.bz2       zshparam.1.bz2
mu-cfind.1.bz2     zshroadmap.1.bz2
mu-extract.1.bz2   zshrcpsys.1.bz2
mu-fields.1.bz2    zshzftsys.1.bz2
mu-find.1.bz2      zshzle.1.bz2
mu-help.1.bz2      zsoelim.1.bz2
mu-index.1.bz2     zstd.1.bz2
mu-info.1.bz2      zstdcat.1.bz2
mu-init.1.bz2      zstdgrep.1.bz2
multig.1.bz2       zstdless.1.bz2
mu-mkdir.1.bz2     zstdmt.1.bz2
munge.1.bz2        zvbi-chains.1.bz2
mupdf.1.bz2        zvbi.1.bz2
mu-remove.1.bz2    zvbi-ntsc-cc.1.bz2
aeavetisyan@dk4n65 ~ $ ls /usr/share/man/man1
```

Рис. 3: Изучение содержимого папки

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 (рис. fig. 4).

```
Открыть  + *122.sh Сохранить  ⌵
report.md x *122.sh
1 #! /bin/bash
2
3 a=$1
4 if test -f "/usr/share/man/man1/$a.1.gz"
5 then less /usr/share/man/man1/$a.1.gz
6 else
7 echo "There is no such command"
8 fi
```

Рис. 4: Код программы

```
#!/bin/bash

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

Проверяю работу командного файла (рис. fig. 5).

```
[evdvorkina@evdvorkina ~]$ ./122.sh ls
[evdvorkina@evdvorkina ~]$
```

Рис. 5: Исполнение программы

Командный файл работает так же, как и команда `man`, открывает справку по указанной утилите (рис. fig. 6).

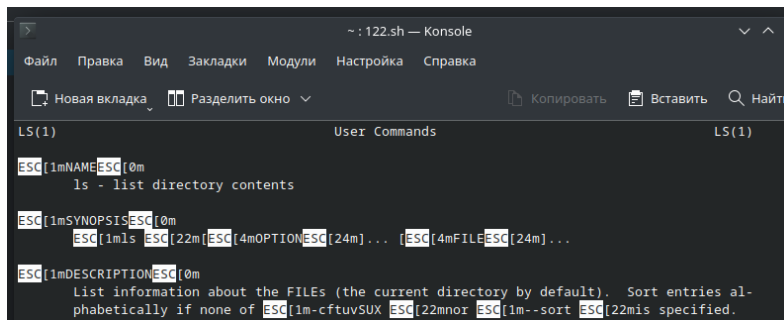


Рис. 6: Результат работы программы

Создаю файл для кода третьей программы, пишу программу и проверяю ее работу (рис. fig. 7).

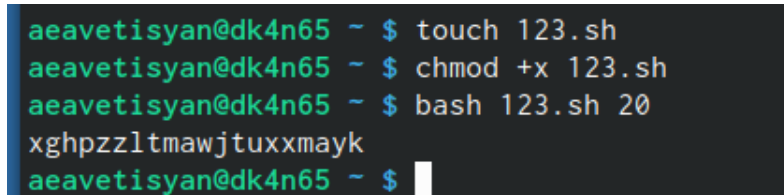


Рис. 7: Создание и исполнение файла

Используя встроенную переменную `$RANDOM`, пишу командный файл, генерирующий случайную последовательность букв латинского алфавита. Т.к. `$RANDOM` выдаёт псевдослучайные числа в диапазоне от 0 до 32767, ввожу ограничения так, чтобы была генерация чисел от 1 до 26 (рис. fig. 8).

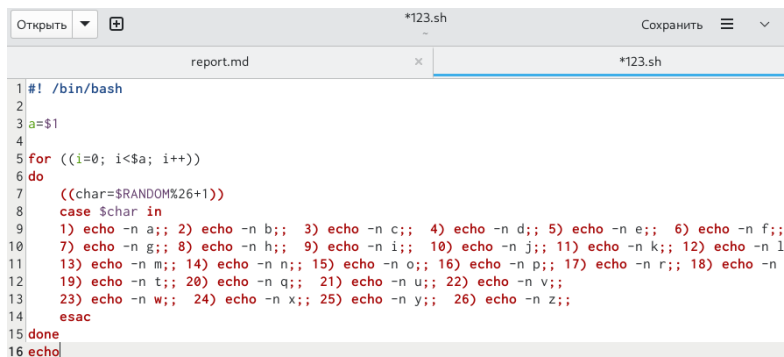


Рис. 8: Код программы

```
#!/bin/bash
```

```
a=$1
```

```
for ((i=0; i<$a; i++))  
do
```

```

((char=$RANDOM%26+1))
case $char in
  1) echo -n a;; 2) echo -n b;; 3) echo -n c;; 4) echo -n d;; 5) echo -n
e;; 6) echo -n f;;
  7) echo -n g;; 8) echo -n h;; 9) echo -n i;; 10) echo -n j;; 11) echo -
n k;; 12) echo -n l;;
  13) echo -n m;; 14) echo -n n;; 15) echo -n o;; 16) echo -n p;; 17) echo
-n r;; 18) echo -n s;;
  19) echo -n t;; 20) echo -n q;; 21) echo -n u;; 22) echo -n v;;
  23) echo -n w;; 24) echo -n x;; 25) echo -n y;; 26) echo -n z;;
esac
done
echo

```

5 Выводы

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

6 Ответы на контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `1 while [$1 != "exit"]`

В данной строчке допущены следующие ошибки: не хватает пробелов после первой скобки [и перед второй скобкой] выражение \$1 необходимо взять в "", потому что эта переменная может содержать пробелы. Таким образом, правильный вариант должен выглядеть так: `while ["$1" != "exit"]`

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

Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами: Первый: `VAR1="Hello," VAR2=" World" VAR3="$VAR1$VAR2" echo "$VAR3"`
 Результат: Hello, World
 Второй: `VAR1="Hello," VAR1+=" World" echo "$VAR1"`
 Результат: Hello, World

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

Команда seq в Linux используется для генерации чисел от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT. Параметры: seq LAST: если задан только один аргумент, он создает числа от 1 до LAST с шагом шага, равным 1. Если LAST меньше 1, значение is не выдает. seq FIRST LAST: когда заданы два аргумента, он генерирует числа от FIRST до LAST с шагом 1, равным 1. Если LAST меньше FIRST, он не выдает никаких выходных данных. seq FIRST INCREMENT LAST: когда заданы три аргумента, он генерирует числа от FIRST до LAST на шаге INCREMENT. Если LAST меньше, чем FIRST, он не производит вывод. seq -f «FORMAT» FIRST INCREMENT LAST: эта команда используется для генерации последовательности в форматированном виде. FIRST и

INCREMENT являются необязательными. seq -s «STRING» ПЕРВЫЙ ВКЛЮЧЕНО: Эта команда используется для STRING для разделения чисел. По умолчанию это значение равно /n. FIRST и INCREMENT являются необязательными. seq -w FIRST INCREMENT LAST: эта команда используется для выравнивания ширины путем заполнения начальными нулями. FIRST и INCREMENT являются необязательными.

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

Результатом данного выражения $\$((10/3))$ будет 3, потому что это целочисленное деление без остатка.

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

Отличия командной оболочки zsh от bash: В zsh более быстрое автодополнение для cd с помощью Tab В zsh существует калькулятор zcalc, способный выполнять вычисления внутри терминала В zsh поддерживаются числа с плавающей запятой В zsh поддерживаются структуры данных «хэш» В zsh поддерживается раскрытие полного пути на основенеполных данных В zsh поддерживается замена части пути В zsh есть возможность отображать разделенный экран, такой же как разделенный экран vim

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

`for ((a=1; a <= LIMIT; a++))` синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать \$ перед переменными ().

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

Преимущества и недостатки скриптового языка bash:

- Один из самых распространенных и ставится по умолчанию в большинстве дистрибутивах Linux, MacOS
- Удобное перенаправление ввода/вывода
- Большое количество команд для работы с файловыми системами Linux
- Можно писать собственные скрипты, упрощающие работу в Linux Недостатки скриптового языка bash:
- Дополнительные библиотеки других языков позволяют выполнить больше действий
- Bash не является языком общего назначения
- Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на скорости выполнения этого скрипта
- Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий

Список литературы