

Отчет по лабораторной работе №14

Операционные системы

Калашникова Ольга Сергеевна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Выводы	14
5	Ответы на контрольные вопросы	15

Список иллюстраций

3.1	Создание и исполнение файла	8
3.2	Код программы	9
3.3	Изучение содержимого папки	10
3.4	Код программы	11
3.5	Исполнение программы	11
3.6	Результат работы программы	12
3.7	Создание и исполнение файла	12
3.8	Код программы	13

Список таблиц

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.

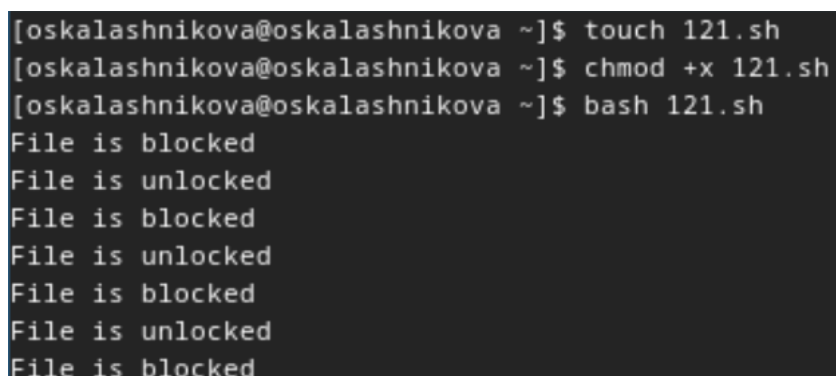
2 Задание

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 Выполнение лабораторной работы

Создаю командный файл для первой программы, пишу ее, проверяю ее работу (рис. fig. 3.1).



```
[oskalashnikova@oskalashnikova ~]$ touch 121.sh
[oskalashnikova@oskalashnikova ~]$ chmod +x 121.sh
[oskalashnikova@oskalashnikova ~]$ bash 121.sh
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
```

Рис. 3.1: Создание и исполнение файла

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 (рис. fig. 3.2).


```
#!/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
```

Рис. 3.2: Код программы

```
#!/bin/bash

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

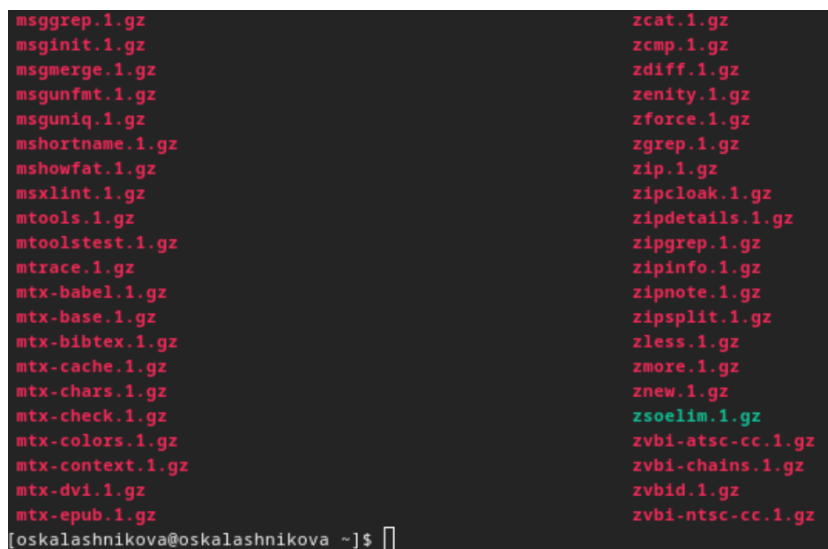
while test -f "$lockfile"
do
    if flock -n ${fn}
    then
```

```

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. 3.3).



```

msggrep.1.gz      zcat.1.gz
msginit.1.gz      zcmp.1.gz
msgmerge.1.gz     zdiff.1.gz
msgunfmt.1.gz     zenity.1.gz
msguniq.1.gz      zforce.1.gz
mshortname.1.gz   zgrep.1.gz
mshowfat.1.gz     zip.1.gz
msxlint.1.gz      zipcloak.1.gz
mtools.1.gz       zipdetails.1.gz
mtoolstest.1.gz   zipgrep.1.gz
mtrace.1.gz       zipinfo.1.gz
mtx-babel.1.gz    zipnote.1.gz
mtx-base.1.gz     zipsplit.1.gz
mtx-bibtex.1.gz   zless.1.gz
mtx-cache.1.gz    zmore.1.gz
mtx-chars.1.gz    znew.1.gz
mtx-check.1.gz    zsoelim.1.gz
mtx-colors.1.gz   zvbi-atsc-cc.1.gz
mtx-context.1.gz  zvbi-chains.1.gz
mtx-dvi.1.gz      zvid.1.gz
mtx-epub.1.gz     zvbi-ntsc-cc.1.gz
[oskalashnikova@oskalashnikova ~]$

```

Рис. 3.3: 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. 3.4).

```
#!/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
```

Рис. 3.4: 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. 3.5).

```
[oskalashnikova@oskalashnikova ~]$ touch 122.sh
[oskalashnikova@oskalashnikova ~]$ chmod +x 122.sh
[oskalashnikova@oskalashnikova ~]$ ./122.sh ls
[oskalashnikova@oskalashnikova ~]$
```

Рис. 3.5: Исполнение программы

Командный файл работает так же, как и команда man, открывает справку по указанной утилите (рис. fig. 3.6).

```

ESC[1m--authorESC[0m
with ESC[1m-lESC[22m, print the author of each file

ESC[1m-bESC[22m, ESC[1m--escapeESC[0m
print C-style escapes for nongraphic characters

ESC[1m--block-sizeESC[22m=ESC[4mSIZEESC[0m
with ESC[1m-lESC[22m, scale sizes by SIZE when pri
nting them;
e.g., '--block-size=M'; see SIZE format below

ESC[1m-BESC[22m, ESC[1m--ignore-backupsESC[0m
do not list implied entries ending with ~

ESC[1m-c ESC[22mwith ESC[1m-ltESC[22m: sort by, and show
, ctime (time of last
change of file status information); with ESC[1m-l
ESC[22m: show
/usr/share/man/man1/ls.1.gz

```

Рис. 3.6: Результат работы программы

Создаю файл для кода третьей программы, пишу программу и проверяю ее работу (рис. fig. 3.7).

```

[oskalashnikova@oskalashnikova ~]$ touch 123.sh
[oskalashnikova@oskalashnikova ~]$ chmod +x 123.sh
[oskalashnikova@oskalashnikova ~]$ bash 123.sh 1
p
[oskalashnikova@oskalashnikova ~]$ bash 123.sh 11
hsuaggkdgzz
[oskalashnikova@oskalashnikova ~]$ bash 123.sh 20
ibeqnoirbgzhjhjbndoj
[oskalashnikova@oskalashnikova ~]$ 

```

Рис. 3.7: 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. 3.8).

```

#!/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

```

Рис. 3.8: 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;;
```

```
        7) echo -n g;; 8) echo -n h;; 9) echo -n i;; 10) echo -n j;; 11) echo -n k;
```

```
        13) echo -n m;; 14) echo -n n;; 15) echo -n o;; 16) echo -n p;; 17) echo -n r
```

```
        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
```

4 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.

5 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 основе неполных 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