

Отчёт по лабораторной работе №12

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

Аскеров Александр Эдуардович

Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Контрольные вопросы	11
4	Вывод	14

Список иллюстраций

2.1	Создание командного файла один, право доступа и запуск	5
2.2	Программа один	6
2.3	Содержимое каталога /usr/share/man/man1	7
2.4	Создание командного файла два, право доступа и запуск	7
2.5	Результат работы	8
2.6	Программа два	9
2.7	Создание командного файла три, право доступа и запуск	9
2.8	Программа три	10

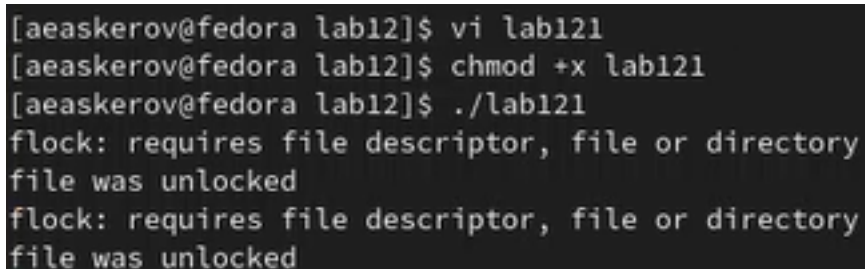
1 Цель работы

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. Напишем 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). Запустим 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. Доработаем программу так, чтобы имелась возможность взаимодействия трёх и более процессов.

Создадим командный файл, напомним в нём программу, дадим файлу право на исполнение, запустим его.



```
[aeaskerov@fedora lab12]$ vi lab121
[aeaskerov@fedora lab12]$ chmod +x lab121
[aeaskerov@fedora lab12]$ ./lab121
flock: requires file descriptor, file or directory
file was unlocked
flock: requires file descriptor, file or directory
file was unlocked
```

Рис. 2.1: Создание командного файла, права доступа и запуск

Приведём саму программу.

```

lockfile="./locking.file"

exec {fn}>"$lockfile"
if test -f "$lockfile"
then
    while [ 1 != 0 ]
    do
        if flock -n ${fn}
        then
            echo "file was locked"
            sleep 4
            echo "unlocking"
            flock -u ${fn}

        else
            echo "file was unlocked"
            sleep 3
        fi
    done
fi

```

Рис. 2.2: Программа один

2. Реализуем команду `man` с помощью командного файла. Изучим 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`.

Изучим содержимое каталога `/usr/share/man/man1`.

```
[aeaskerov@fedora lab12]$ cd /usr/share/man/man1
[aeaskerov@fedora man1]$ ls
:~.1.gz
'~.1.gz'
a2ping.1.gz
ab.1.gz
```

Рис. 2.3: Содержимое каталога /usr/share/man/man1

Создадим командный файл, напишем в нём программу, дадим файлу право на исполнение, запустим его. Например, посмотрим информацию о команде ls.

```
[aeaskerov@fedora lab12]$ vi lab122
[aeaskerov@fedora lab12]$ chmod +x lab122
[aeaskerov@fedora lab12]$ ./lab122 -c ls
```

Рис. 2.4: Создание командного файла два, право доступа и запуск

Покажем, результат работы командного файла.

```
aeaskerov@fedora:~/work/study/2022-2023/Операционные с...
LS(1) User Commands LS(1)
ESC[1mNAMEESC[0m
ls - list directory contents

ESC[1mSYNOPSISESC[0m
ESC[1mls ESC[22m[ESC[4mOPTIONESC[24m]... [ESC[4mFILEESC[24m]...

ESC[1mDESCRIPTIONESC[0m
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of ESC[1m-cftuvSUX ESC[22mnor ESC[1m-
-sort ESC[22mis speci-
fied.

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

ESC[1m-aESC[22m, ESC[1m--allESC[0m
do not ignore entries starting with .

ESC[1m-AESC[22m, ESC[1m--almost-allESC[0m
do not list implied . and ..

/usr/share/man/man1/ls.1.gz
```

Рис. 2.5: Результат работы

Приведём саму программу.


```

command=""

while getopts :c: opt
do
case $opt in
        c) command="$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

```

Рис. 2.6: Программа два

- Используя встроенную переменную \$RANDOM, напомним командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтём, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

Создадим командный файл, напомним в нём программу, дадим файлу право на исполнение, запустим его. Видим, что последовательности генерируются.

```

[aeaskerov@fedora lab12]$ vi lab123
[aeaskerov@fedora lab12]$ chmod +x lab123
[aeaskerov@fedora lab12]$ ./lab123
eajj
[aeaskerov@fedora lab12]$ ./lab123
iabj
[aeaskerov@fedora lab12]$ ./lab123
dabih

```

Рис. 2.7: Создание командного файла три, право доступа и запуск

Приведём саму программу.

```
echo $RANDOM | tr '0-9' 'a-zA-Z'
```

Рис. 2.8: Программа три

3 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке:

```
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="VAR1VAR2" 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` 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

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 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.

4 Вывод

Изучены основы программирования в оболочке ОС UNIX. Приобретён навык написания более сложных командных файлов с использованием логических управляющих конструкций и циклов.