

Отчёт по лабораторной работе №14

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

Джаллох Ишмаил

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
3.1	командный файл, реализующий упрощённый механизм семафоров	7
3.2	Реализовать команду tap с помощью командного файла	8
3.3	написать командный файл, генерирующий случайную последовательность букв латинского алфавита.	10
4	Выводы	12
5	Ответы на контрольные вопросы	13
	Список литературы	15

Список иллюстраций

3.1	упрощённый механизм семафоров (код)	7
3.2	результаты кода	8
3.3	ls /usr/share/man/man1	9
3.4	командный файл man	9
3.5	проверка командного файла man	10
3.6	проверка командного файла man	10
3.7	командный файл, генерирующий случайную последовательность букв	10
3.8	запуск скрипта	11

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 Задание

1. Написать командный файл, реализующий упрощённый механизм семафоров.
2. Реализовать команду `map` с помощью командного файла.
3. Используя встроенную переменную `$RANDOM`, написать командный файл, генерирующий случайную последовательность букв латинского алфавита.

3 Выполнение лабораторной работы

3.1 командный файл, реализующий упрощённый механизм семафоров

Чтобы создать данный командный файл, я создал новый файл и написал в нем некоторый скрипт. Он устанавливает переменную `lockfile` для пути к файлу блокировки, открывает файл для записи и назначает ему дескриптор файла. Далее входит в цикл, который выполняется, пока файл блокировки существует. Пытается получить эксклюзивную блокировку для файла. Если это удастся, выводит “file locked”, ждёт 5 секунд а затем выводит “file unlocked”:

A screenshot of a terminal window titled "lab14_file1.sh". The window contains a shell script with 17 lines of code. The script sets a lockfile path, uses exec to open it for writing, and enters a while loop that checks if the lockfile exists. Inside the loop, it attempts to acquire an exclusive lock with flock -n. If successful, it prints "File is locked", sleeps for 5 seconds, prints "File is unlocked", and releases the lock with flock -u. If the lock cannot be acquired, it prints "File is locked" and sleeps for 5 seconds. The loop ends with a 'done' statement.

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

Рис. 3.1: упрощённый механизм семафоров (код)

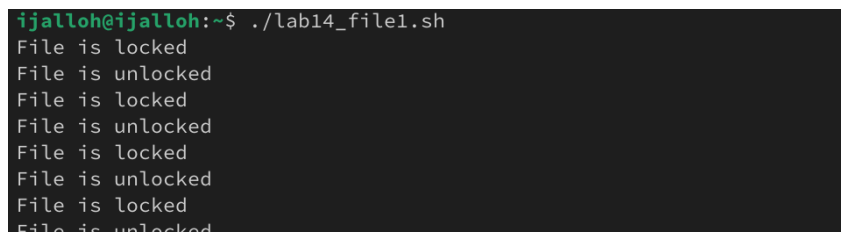
```
lockfile="./lock.file"
```

```
exec {fn}>$lockfile
```

```
while test -f "$lockfile"
```

```
do
if flock -n ${fn}
then
    echo "File is locked"
    sleep 5
    echo "File is unlocked"
    flock -u ${fn}
else
    echo "File is locked"
    sleep 5

fi
done
```



```
ijalloheijalloh:~$ ./lab14_file1.sh
File is locked
File is unlocked
File is locked
File is unlocked
File is locked
File is unlocked
File is locked
File is unlocked
```

Рис. 3.2: результаты кода

3.2 Реализовать команду man с помощью командного файла

Я изучил содержимое каталога /usr/share/man/man1. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд:

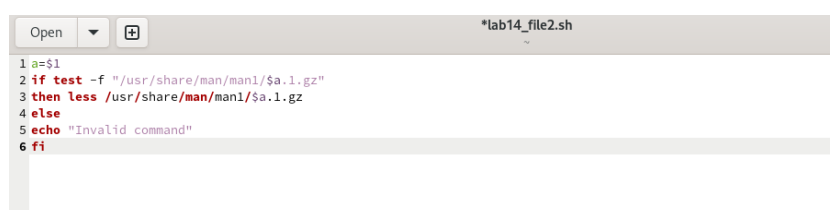

```

zip.1.gz
zipcloak.1.gz
zipdetails.1.gz
zipgrep.1.gz
zipinfo.1.gz
zipnote.1.gz
zipsplit.1.gz
zless.1.gz
zmore.1.gz
znew.1.gz
zsoelim.1.gz
zstd.1.gz
zstdcat.1.gz
zstdgrep.1.gz
zstdless.1.gz
zvbi-atsc-cc.1.gz
zvbi-chains.1.gz
zvbid.1.gz
zvbi-ntsc-cc.1.gz

```

Рис. 3.3: ls /usr/share/man/man1

Потом я создал файл и в нем написал скрипт реализующий команды man. Он принимает аргумент \$1, проверяет существование файла в /usr/share/man/man1, и если файл существует, использует less для отображения содержимого сжатой страницы руководства. Если файл не существует, выводит “invalid command”:



```

1 a=$1
2 if test -f "/usr/share/man/man1/$a.1.gz"
3 then less /usr/share/man/man1/$a.1.gz
4 else
5 echo "Invalid command"
6 fi

```

Рис. 3.4: командный файл man

```

a=$1
if test -f "/usr/share/man/man1/$a.1.gz"
then less /usr/share/man/man1/$a.1.gz
else
echo "Invalid command"
fi

```

```

ijalloh@ijalloh:~$ gedit lab14_file2.sh
ijalloh@ijalloh:~$ chmod +x lab14_file2.sh
ijalloh@ijalloh:~$ ./lab14_file2.sh
Invalid command
ijalloh@ijalloh:~$ ./lab14_file2.sh ls
ijalloh@ijalloh:~$

```

Рис. 3.5: проверка командного файла man

```

ESC[4mL ESC[24m(1) User Commands
ESC[4mL ESC[24m(1)

ESC[1mNAME ESC[0m
ls - list directory contents

ESC[1mSYNOPSIS ESC[0m
ESC[1m ls ESC[22m[ESC[4mOPTION ESC[24m]... [ESC[4mFILE ESC[24m]...

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

```

Рис. 3.6: проверка командного файла man

3.3 написать командный файл, генерирующий случайную последовательность букв латинского алфавита.

Я написал скрипт который генерирует случайное число используя \$RANDOM, а затем с помощью tr заменяет каждую цифру на букву от 'a-z' и 'A-Z':

```

Open [~] *lab14_file3.sh
1 echo $RANDOM | tr '0-9' 'a-zA-Z'

```

Рис. 3.7: командный файл, генерирующий случайную последовательность букв

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

```
ijalloh@ijalloh:~$ ./lab14_file3.sh
begjf
ijalloh@ijalloh:~$ ./lab14_file3.sh
cbacd
ijalloh@ijalloh:~$ ./lab14_file3.sh
cgdbf
ijalloh@ijalloh:~$ ./lab14_file3.sh
ceja
ijalloh@ijalloh:~$
```

Рис. 3.8: запуск скрипта

4 Выводы

При выполнении данной работы я научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 Ответы на контрольные вопросы

1. 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. Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами: Первый: `VAR1=“Hello,” VAR2=“ World” VAR3=“VAR1VAR2”`
`echo “VAR3” : Hello, World : VAR1 = “Hello,”VAR1+ =`
`“World”echo“VAR1”` Результат: Hello, World
3. 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. Результатом данного выражения $\$(10/3)$ будет 3, потому что это целочисленное деление без остатка.
5. 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. for ((a=1; a <= LIMIT; a++)) синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать \$ перед переменными ().
7. 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

Архитектура ЭВМ