

Отчёт по лабораторной работе №14

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

Дауд Амжад

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	9
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 создала новый файл и написала 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”, ждет 5 секунд а затем выводит “file unlocked”:

A screenshot of a terminal window titled "lab14_file1.sh". The window contains a shell script with 16 lines of code. The script sets a lockfile path, uses flock to attempt exclusive locking, and prints status messages with a 5-second delay. The code is as follows:

```
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
```

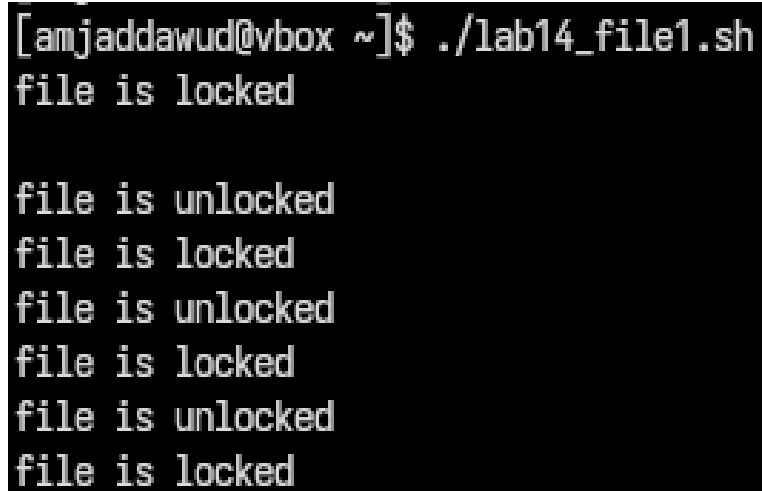
Рис. 3.1: упрощённый механизм семафоров (код)

```
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
```

A terminal window with a black background and white text. The prompt is [amjaddawud@vbox ~]\$ and the command is ./lab14_file1.sh. The output shows a sequence of 'file is locked' and 'file is unlocked' messages. The first 'file is locked' message is on the same line as the command. There is a blank line after the first 'file is locked' message. The sequence of messages is: 'file is locked', blank line, 'file is unlocked', 'file is locked', 'file is unlocked', 'file is locked', 'file is unlocked', 'file is locked'.

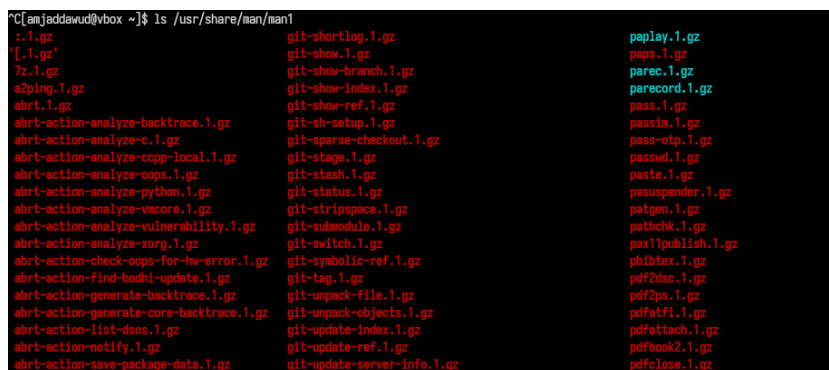
```
[amjaddawud@vbox ~]$ ./lab14_file1.sh
file is locked

file is unlocked
file is locked
file is unlocked
file is locked
file is unlocked
file is locked
```

Рис. 3.2: результаты кода

3.2 Реализовать команду man с помощью командного файла

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



```
^C[anjaddawud@vbox ~]$ ls /usr/share/man/man1
.:1.gz
[1.gz
7z.1.gz
a2ping.1.gz
abrt.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
abrt-action-analyze-ops.1.gz
abrt-action-analyze-python.1.gz
abrt-action-analyze-vmcore.1.gz
abrt-action-analyze-vulnerability.1.gz
abrt-action-analyze-xorg.1.gz
abrt-action-check-ops-for-hw-error.1.gz
abrt-action-find-bodhi-update.1.gz
abrt-action-generate-backtrace.1.gz
abrt-action-generate-core-backtrace.1.gz
abrt-action-list-dsos.1.gz
abrt-action-notify.1.gz
abrt-action-save-package-data.1.gz
git-shortlog.1.gz
git-show.1.gz
git-show-branch.1.gz
git-show-index.1.gz
git-show-ref.1.gz
git-sh-setup.1.gz
git-sparse-checkout.1.gz
git-stage.1.gz
git-stash.1.gz
git-status.1.gz
git-strip-space.1.gz
git-submodule.1.gz
git-switch.1.gz
git-symbolic-ref.1.gz
git-tag.1.gz
git-unpack-file.1.gz
git-unpack-objects.1.gz
git-update-index.1.gz
git-update-ref.1.gz
git-update-server-info.1.gz
paplay.1.gz
paps.1.gz
parec.1.gz
parecord.1.gz
pass.1.gz
passim.1.gz
passim.1.gz
passwd.1.gz
paste.1.gz
pasuspender.1.gz
patgen.1.gz
pathchk.1.gz
pax11publish.1.gz
pbibtex.1.gz
pdf2dsc.1.gz
pdf2ps.1.gz
pdfcat.1.gz
pdfattach.1.gz
pdfbook2.1.gz
pdfclose.1.gz
```

Рис. 3.3: `ls /usr/share/man/man1`

Потом я создала файл и в нем написала 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”:



```
lab14_file2.sh
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

```

```

[amjaddawud@vbox ~]$ gedit lab14_file2.sh
(gedit:13371): Gtk-WARNING **: 17:19:51.454: Calling org.freedesktop.portal.Inhibit.InhibitMethod: No such interface "org.freedesktop.portal.Inhibit" on object at path /org/freedesktop/portal/desktop/Inhibit
[amjaddawud@vbox ~]$ chmod +x lab14_file2.sh
[amjaddawud@vbox ~]$ ./lab14_file2.sh
Invalid command
[amjaddawud@vbox ~]$ ./lab14_file2.sh ls

```

Рис. 3.5: проверка командного файла man

```

ESC[4mLSESC[24m(1) User Commands
ESC[4mLSESC[24m(1)
ESC[1mNAMEESC[0m
ls - list directory contents
ESC[1mSYNOPSISESC[0m
ESC[1mls ESC[22m[ESC[4mOPTIONESC[24m]... [ESC[4mFILEESC[24m]...
ESC[1mDESCRIPTIONESC[0m

```

Рис. 3.6: проверка командного файла man

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

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

```

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'
```

```
[amjaddawud@vbox ~]$ ./lab14_file3.sh
gege
[amjaddawud@vbox ~]$ ./lab14_file3.sh
jjhf
[amjaddawud@vbox ~]$ ./lab14_file3.sh
bedbb
```

Рис. 3.8: запуск скрипта

4 Выводы

При выполнении данной работы я научилась 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