

Отчет по лабораторной работе №12

Операционные системы

Ничипорова Елена Дмитриевна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	17
	Список литературы	18

Список иллюстраций

2.1	Создание файла	6
2.2	Скрипт №1	7
2.3	Проверка скрипта №1	7
2.4	Измененный скрипт №1	8
2.5	Измененный скрипт №1	9
2.6	проверка работы скрипта №1	9
2.7	Реализация команды map	10
2.8	создание нового файла	10
2.9	скрипт №2	10
2.10	проверка работы скрипта №2	11
2.11	проверка работы скрипта №2	11
2.12	проверка работы скрипта №2	12
2.13	проверка работы скрипта №2	13
2.14	скрипт №3	13
2.15	проверка работы скрипта №3	14

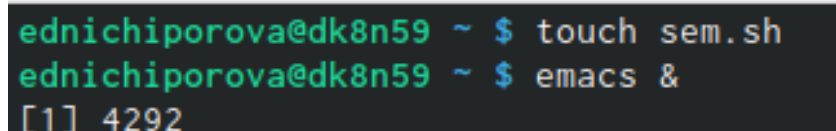
Список таблиц

1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Выполнение лабораторной работы

1. Написала 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. Для данной задачи я создала новый файл (рис. 2.1) и написала соответствующий скрипт (рис. 2.2)

A terminal window with a dark background and light green text. The prompt is 'ednichiporova@dk8n59 ~ \$'. The first command is 'touch sem.sh' and the second is 'emacs &'. The output shows '[1] 4292' on the next line.

```
ednichiporova@dk8n59 ~ $ touch sem.sh
ednichiporova@dk8n59 ~ $ emacs &
[1] 4292
```

Рис. 2.1: Создание файла

```
#!/bin/bash
t1=$1
t2=$2
s1=$(date +%s)
s2=$(date +%s)
((t=s2-s1))
while ((t<t1))
do
    echo "ожидание"
    sleep 1
    s2=$(date +%s)
    ((t=s2-s1))
done
s1=$(date +%s)
s2=$(date +%s)
((t=s2-s1))
while ((t<t2))
do
    echo "выполнение"
    sleep 1
    s2=$(date +%s)
    ((t=s2-s1))
done
```

Рис. 2.2: Скрипт №1

- Далее я проверила работу написанного скрипта, перед этим я предоставила право на исполнение данного файла командой “chmod +x sem.sh”. Скрипт работает корректно(рис. 2.3)

```
ednichiporova@dk8n59 ~ $ chmod +x sem.sh
ednichiporova@dk8n59 ~ $ ./sem.sh 2 3
ожидание
ожидание
выполнение
выполнение
выполнение
ednichiporova@dk8n59 ~ $
```

Рис. 2.3: Проверка скрипта №1

- После этого я изменила скрипт так, чтобы его можно было выполнять в нескольких терминалах (рис. 2.4) (рис. 2.5) и проверила его работу, но у

меня не получилось проверить скрипт, так как было отказано в доступе (рис. 2.6)

```
#!/bin/bash
function ogidania
{
s1=$(date +%s")
s2=$(date +%s")
((t=$s2-$s1))
while ((t<t1))
do
    echo "Ожидание"
    sleep 1
    s2=$(date +%s")
    ((t=$s2-$s1))
done
}
function vipolnenie
{
s1=$(date +%s")
s2=$(date +%s")
((t=$s2-$s1))
while ((t<t2))
do
    echo "Выполнение"
    sleep 1
    s2=$(date +%s" )
    ((t=$s2-$s1))
done
}
t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then ogidanie
    fi
    if [ "$command" == "Выполнение" ]
```

Рис. 2.4: Измененный скрипт №1


```

Edit Options Buffers Tools Sn-Script Help
|  then ogidanie
    fi
    if [ "$command" == "Выполнение" ]
    then vipolnenie
    fi
    echo "следующее действие: "
    read command
done

```

Рис. 2.5: Измененный скрипт №1

```

ednichiporova@dk8n59 ~ $ chmod +x sem.sh
ednichiporova@dk8n59 ~ $ ./sem.sh 2 3 Ожидание > /dev/pts/1 &
[2] 10165
ednichiporova@dk8n59 ~ $ bash: /dev/pts/1: Отказано в доступе
./sem.sh 2 3 Ожидание > /
[2]+  Выход 1          ./sem.sh 2 3 Ожидание > /dev/pts/1

```

Рис. 2.6: проверка работы скрипта №1

2. Реализовала команду `man` с помощью командного файла. Изучила содержимое каталога `/usr/share/man/man1`. (рис. 2.7) 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`.

[illegible]

- Для данной задачи я создала новый файл (рис. 2.8) и написала соответствующий скрипт(рис. 2.9)

```
#!/bin/bash
a=$1
if [ -f /usr/share/man/man1/$a.1.bz2 ]
then
    bunzip2 -c /usr/share/man/man1/$1.1.bz2 | less
else
    echo "справки по данной команде нет"
fi
```

- Проверила работу скрипта, предварительно добавив право на исполнение файла. Скрипт работает корректно (рис. 2.10) (рис. 2.11) (рис. 2.12) (рис. 2.13)

```

ednichiporova@dk8n59 ~ $ ./man.sh mkdir
ednichiporova@dk8n59 ~ $
ednichiporova@dk8n59 ~ $ ./man.sh rm
ednichiporova@dk8n59 ~ $ ./man.sh cat

```

Рис. 2.10: проверка работы скрипта №2

```

.\ DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH MKDIR "1" "March 2020" "GNU coreutils 8.32" "User Commands"
.SH NAME
mkdir \- make directories
.SH SYNOPSIS
.B mkdir
[\FI,\OPTION]\FR... \FI,\DIRECTORY\FR...
.SH DESCRIPTION
.\ Add any additional description here
.PP
Create the DIRECTORY(ies), if they do not already exist.
.PP
Mandatory arguments to long options are mandatory for short options too.
.TP
\FB\-m\FR, \FB\-mode\FR=\FI,\MODE\FR
set file mode (as in chmod), not a=rx \-umask
.TP
\FB\-p\FR, \FB\-parents\FR
no error if existing, make parent directories as needed
.TP
\FB\-v\FR, \FB\-verbose\FR
print a message for each created directory
.TP
\FB\-Z\FR
set SELinux security context of each created directory
to the default type
.TP
\FB\-context\FR[=\FI,\CTX]\FR
like \FB\-Z\FR, or if CTX is specified then set the SELinux
or SMACK security context to CTX
.TP
\FB\-help\FR
display this help and exit
.TP
\FB\-version\FR
output version information and exit
.SH AUTHOR
Written by David MacKenzie.
.SH "REPORTING BUGS"
GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
.br
Report any translation bugs to <https://translationproject.org/team/>
.SH "SEE ALSO"
mkdir(2)
.br
Full documentation <https://www.gnu.org/software/coreutils/mkdir>
.br
or available locally via: info \aq(coreutils) mkdir invocation\aq
.PP
Packaged by Gentoo (8.32-r1 (p0))
.br
Copyright © 2020 Free Software Foundation, Inc.

```

Рис. 2.11: проверка работы скрипта №2

```

.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH RM "1" "March 2020" "GNU coreutils 8.32" "User Commands"
.SH NAME
rm \- remove files or directories
.SH SYNOPSIS
.B rm
[\fI\,OPTION\ \fR]... [\fI\,FILE\ \fR]...
.SH DESCRIPTION
This manual page
documents the GNU version of
.BR rm .
.B rm
removes each specified file. By default, it does not remove
directories.
.P
If the \fI\-I\fR or \fI\--interactive=once\fR option is given,
and there are more than three files or the \fI\-r\fR, \fI\-R\fR,
or \fI\--recursive\fR are given, then
.B rm
prompts the user for whether to proceed with the entire operation. If
the response is not affirmative, the entire command is aborted.
.P
Otherwise, if a file is unwritable, standard input is a terminal, and
the \fI\-f\fR or \fI\--force\fR option is not given, or the
\fI\-i\fR or \fI\--interactive=always\fR option is given,
.B rm
prompts the user for whether to remove the file. If the response is
not affirmative, the file is skipped.
.SH OPTIONS
.PP
Remove (unlink) the FILE(s).
.TP
\fB\--force\fR, \fB\--force\fR
ignore nonexistent files and arguments, never prompt
.TP
\fB\--interactive\fR
prompt before every removal
.TP
\fB\--interactive\fR
prompt once before removing more than three files, or
when removing recursively; less intrusive than \fB\--i\fR,
while still giving protection against most mistakes
.TP
\fB\--interactive\fR[=\fI\,WHEN\ \fR]
prompt according to WHEN: never, once (\fB\--I\fR), or
always (\fB\--i\fR); without WHEN, prompt always
.TP
\fB\--no-preserve-root\fR
do not treat '/' specially (the default)

```

Рис. 2.12: проверка работы скрипта №2

```

.TH CAT "1" "March 2020" "GNU coreutils 8.32" "User Commands"
.SH NAME
cat \- concatenate files and print on the standard output
.SH SYNOPSIS
.B cat
[\fI,\fOOPTION\fR... [\fI,\fOFILE\fR...
.SH DESCRIPTION
.\" Add any additional description here
.PP
Concatenate FILE(s) to standard output.
.PP
With no FILE, or when FILE is \-, read standard input.
.TP
\FB\-\A\FR, \FB\-\show\-\all\FR
equivalent to \FB\-\ET\FR
.TP
\FB\-\b\FR, \FB\-\number\-\nonblank\FR
number nonempty output lines, overrides \FB\-\n\FR
.TP
\FB\-\e\FR
equivalent to \FB\-\vE\FR
.TP
\FB\-\E\FR, \FB\-\show\-\ends\FR
display $ at end of each line
.TP
\FB\-\n\FR, \FB\-\number\FR
number all output lines
.TP
\FB\-\s\FR, \FB\-\squeeze\-\blank\FR
suppress repeated empty output lines
.TP
\FB\-\t\FR
equivalent to \FB\-\vT\FR
.TP
\FB\-\T\FR, \FB\-\show\-\tabs\FR
display TAB characters as ^I
.TP
\FB\-\u\FR
(ignored)
.TP
\FB\-\v\FR, \FB\-\show\-\nonprinting\FR
use ^ and M\ notation, except for LFD and TAB
.TP
\FB\-\help\FR
display this help and exit
.TP
\FB\-\version\FR
output version information and exit
.SH EXAMPLES

```

Рис. 2.13: проверка работы скрипта №2

- Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. Для этого создала новый файл и написала соответствующий скрипт (рис. 2.14)

```

#!/bin/bash
k=$1
for (( i=0; i< $k; i++ ))
do
    (( char=$((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 q;; 18) echo -n r;; 19) echo -n s;; 20) echo -n t;; 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. 2.14: скрипт №3

- Проверила работу данного скрипта, предварительно добавив право на исполнение. Скрипт работает корректно (рис. 2.15)

```
ednichiporova@dk8n59 ~ $ chmod +x random.sh
ednichiporova@dk8n59 ~ $ ./random.sh 5
wzqdu
ednichiporova@dk8n59 ~ $ ./random.sh 12
vbelpzcoqycf
```

Рис. 2.15: проверка работы скрипта №3

- Контрольные вопросы

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 про

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"

Результат: Hello, World

Второй: 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

seq FIRST LAST: когда заданы два аргумента, он генерирует числа от FIRST до LAST

seq FIRST INCREMENT LAST: когда заданы три аргумента, он генерирует числа от FIRST

seq -f «FORMAT» FIRST INCREMENT LAST: эта команда используется для генерации посл

seq -s «STRING» ПЕРВЫЙ ВКЛЮЧЕНО: Эта команда используется для STRING для разделен

seq -w FIRST INCREMENT LAST: эта команда используется для выравнивания ширины путе

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 cdc 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 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,

Скрипты, написанные на bash, нельзя запустить на других операционных системах без

3 Выводы

Изучила основы программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Список литературы