

## Цель работы

---

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

## Задание

---

Научиться писать более сложные командные файлы.

## Выполнение лабораторной работы

---

Написал 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). Для данной задачи я создал файл: sem.sh и написал соответствующий скрипт (рис. -@fig:001)

```
#!/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
```

{ #fig:001 width=70% }

Далее я проверил работу написанного скрипта (команда «./sem.sh 4 7»), предварительно добавив право на исполнение файла (команда «chmod +x sem.sh»). Скрипт работает корректно (рис. -@fig:002)

```

aagoncharov1@aagoncharov1-VirtualBox:~/lab13$ chmod +x prog1.sh
aagoncharov1@aagoncharov1-VirtualBox:~/lab13$ ./prog1.sh 3 9
Ожидание
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
aagoncharov1@aagoncharov1-VirtualBox:~/lab13$

```

{ #fig:002 width=70% }

После этого я изменил скрипт так, чтобы его можно было выполнять в нескольких терминалах и проверил его работу (рис. -@fig:003) (рис. -@fig:004) (рис. -@fig:005) (рис. -@fig:006) (рис. -@fig:007)

```

#!/bin/bash
function waiting
{
    s1=$(date +%s)
    s2=$(date +%s)
    ((t= $s2 - $s1))
    while ((t < t1))
    do
        echo "Ожидание"
        sleep 1
        s2=$(date +%s)
        ((t= $s2 - $s1))
    done
}
function execution
{
    s1=$(date +%s)
    s2=$(date +%s)
    ((t= $s2 - $s1))
    while ((t < t2))
    do
        echo "Выполнение"
        sleep 1
        s2=$(date +%s)
        ((t= $s2 - $s1))
    done
}

```

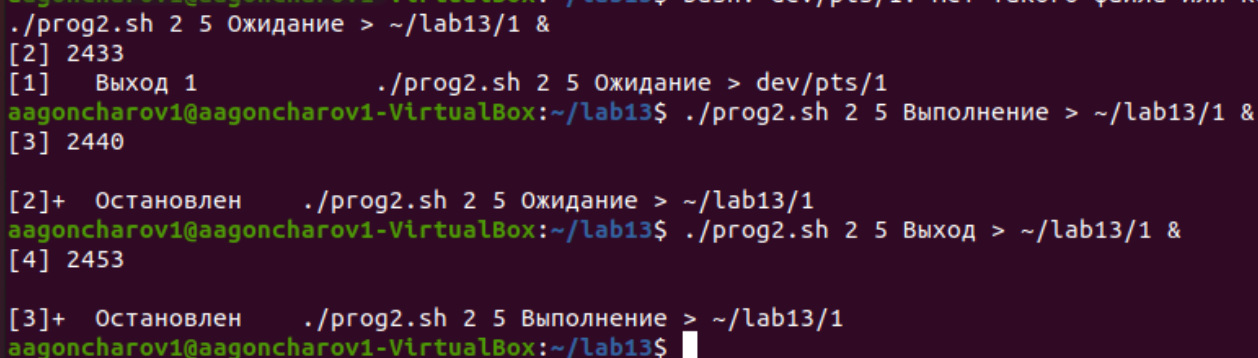
{ #fig:003 width=70% }

```

t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then waiting
    fi
    if [ "$command" == "Выполнение" ]
    then execution
    fi
    echo "Следующее действие: "
    read command
done

```

{ #fig:004 width=70% }



```

./prog2.sh 2 5 Ожидание > ~/lab13/1 &
[2] 2433
[1] Выход 1 ./prog2.sh 2 5 Ожидание > dev/pts/1
aagoncharov1@aagoncharov1-VirtualBox:~/lab13$ ./prog2.sh 2 5 Выполнение > ~/lab13/1 &
[3] 2440

[2]+ Остановлен ./prog2.sh 2 5 Ожидание > ~/lab13/1
aagoncharov1@aagoncharov1-VirtualBox:~/lab13$ ./prog2.sh 2 5 Выход > ~/lab13/1 &
[4] 2453

[3]+ Остановлен ./prog2.sh 2 5 Выполнение > ~/lab13/1
aagoncharov1@aagoncharov1-VirtualBox:~/lab13$

```

{ #fig:005 width=70% }

Реализовал команду man с помощью командного файла. Изучил 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 (рис. -@fig:008)

```

aagoncharov1@aagoncharov1-VirtualBox:~/lab13$ cd /usr/share/man/man1
[4] Завершён ./prog2.sh 2 5 Выход > ~/lab13/1 (рабочий каталог: ~/lab13)
(рабочий каталог: /usr/share/man/man1)
aagoncharov1@aagoncharov1-VirtualBox:/usr/share/man/man1$
aagoncharov1@aagoncharov1-VirtualBox:/usr/share/man/man1$ cd /usr/share/man/man1
aagoncharov1@aagoncharov1-VirtualBox:/usr/share/man/man1$ ls
' [.1.gz'                                mkzftree.1.gz
aa-enabled.1.gz                         mlabel.1.gz
aa-exec.1.gz                           mmcli.1.gz
aconnect.1.gz                          mmd.1.gz
add-apt-repository.1.gz                mmount.1.gz
addr2line.1.gz                         mmove.1.gz
alsabat.1.gz                           more.1.gz
alsactl.1.gz                           mountpoint.1.gz
alsaloop.1.gz                           mousetweaks.1.gz
alsamixer.1.gz                         mpartition.1.gz
alsatplg.1.gz                           mpris-proxy.1.gz
alsaucm.1.gz                           mrd.1.gz
amidi.1.gz                             mren.1.gz
amixer.1.gz                             mscompress.1.gz
apg.1.gz                               msexpand.1.gz
apgbfm.1.gz                             mshortname.1.gz
aplay.1.gz                             mshowfat.1.gz
aplaymidi.1.gz                          mt.1.gz
apport-bug.1.gz                         mt-gnu.1.gz
apport-cli.1.gz                         mtools.1.gz
apport-collect.1.gz                    mtools-test.1.gz

```

{ #fig:008 width=70% }

Для данной задачи я создал файл: man.sh и написал соответствующий скрипт (рис. -@fig:009)

```

#!/bin/bash
c=$1
if [ -f /usr/share/man/man1/$c.1.gz ]
then
    gunzip -c /usr/share/man/man1/$1.1.gz | less
else
    echo "Справки по данной командой нет"
fi

```

{ #fig:009 width=70% }

Далее я проверил работу написанного скрипта (команды «./man.sh ls» и «./man.sh mkdir»), предварительно добавив право на исполнение файла (команда «chmod +x man.sh»). Скрипт работает корректно (рис. -@fig:010) (рис. -@fig:011) (рис. -@fig:012)

 Рисунок 10 { #fig:010 width=70% }

```

.\" DO NOT MODIFY THIS FILE!  It was generated by help2man 1.47.3.
.TH LS "1" "September 2019" "GNU coreutils 8.30" "User Commands"
.SH NAME
ls \- list directory contents
.B ls
[\\fI\\,OPTION\\/\fR]... [\\fI\\,FILE\\/\fR]...
.SH DESCRIPTION
.\" Add any additional description here
.PP
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of \\fB\\-cftuvSUX\\fR nor \\fB\\-\\-sort\\fR is specified.
.PP
Mandatory arguments to long options are mandatory for short options too.
.TP
\\fB\\-a\\fR, \\fB\\-\\-all\\fR
do not ignore entries starting with .
.TP
\\fB\\-A\\fR, \\fB\\-\\-almost\\-all\\fR
do not list implied . and ..
.TP
\\fB\\-\\-author\\fR
with \\fB\\-l\\fR, print the author of each file
.TP
\\fB\\-b\\fR, \\fB\\-\\-escape\\fR
print C\\-style escapes for nongraphic characters
.TP
\\fB\\-\\-block\\-size\\fR=\\fI\\,SIZE\\/\fR
with \\fB\\-l\\fR, scale sizes by SIZE when printing them;
e.g., '\\-\\-block\\-size=M'; see SIZE format below
.TP
\\fB\\-B\\fR, \\fB\\-\\-ignore\\-backups\\fR
do not list implied entries ending with ~
:gzip: ].gz: No such file or directory
.

```

{ #fig:011 width=70% }

```
.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH MKDIR "1" "September 2019" "GNU coreutils 8.30" "User Commands"
.SH NAME
mkdir \- make directories
.SH SYNOPSIS
.B mkdir
[\fI\,OPTION\fR]\fR... [\fI\,DIRECTORY\fR]\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=rwx \- 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
:
```

{ #fig:012 width=70% }

Используя встроенную переменную \$RANDOM, написал командный файл, генерирующий случайную последовательность букв латинского алфавита. Для данной задачи я создал файл: random.sh и написал соответствующий скрипт (рис. -@fig:013)

```
#!/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 l;; 12) echo -n k;; 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
```

{ #fig:013 width=70% }

Далее я проверил работу написанного скрипта (команды «./random.sh 7» и «./random.sh 15»), предварительно добавив право на исполнение файла (команда «chmod +x random.sh») Скрипт работает корректно (рис. -@fig:014)

```

aagoncharov1@aagoncharov1-VirtualBox:~/lab13$
aagoncharov1@aagoncharov1-VirtualBox:~/lab13$ touch random.sh
aagoncharov1@aagoncharov1-VirtualBox:~/lab13$ emacs random.sh
aagoncharov1@aagoncharov1-VirtualBox:~/lab13$ chmod +x random.sh
aagoncharov1@aagoncharov1-VirtualBox:~/lab13$ ./random.sh 15
bchzjmgiiiranv
aagoncharov1@aagoncharov1-VirtualBox:~/lab13$ ./random.sh 21
lyityaipabtevyhlhxcgvd
aagoncharov1@aagoncharov1-VirtualBox:~/lab13$

```

{ #fig:014 width=70% }

## 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 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="\$VAR1\$VAR2" echo "\$VAR3" Результат: Hello, World  
Второй: VAR1="Hello, " VAR1+=" World" echo "\$VAR1" Результат: Hello, World
3. Команда seq в Linux используется для генерации 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 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

## Выводы

---

В ходе выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX, а также научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.