



25.5.2022

Лабораторная работа 12

Манатов Рамазан Русланович
[НАЗВАНИЕ ОРГАНИЗАЦИИ]

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

1) 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 создадим файл `sem.sh` и напишем соответствующий скрипт (рис 1)

```
#!/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 "done"
    sleep 1
    s2=$(date +%s)
    ((t=$s2-$s1))
done
```

Рисунок 1

Далее я проверил работу написанного скрипта предварительно добавив право на выполнение файла (рис2)

```
[rrManatov@fedora ~]$ ./sem.sh 4 7
Ожидание
Ожидание
Ожидание
Ожидание
done
done
done
done
done
```

Рис 2 1 код работает корректно

После этого я изменил скрипт так чтобы его можно было выполнять в нескольких терминал и проверил его работу (рис3) (рис4)

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

```

        echo "done"
        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"]
    then vipolnenie
    fi
    echo "Следующее действие: "
    read command
done

```

Рисунок 2 исправленный код

```

[rrManatov@fedora ~]$ ./sem.sh 2 3 ожидание > /dev/pts/1 &
[2] 3585

```

```

[rrManatov@fedora ~]$ ./sem.sh 3 4 ожидание > /dev/pts/2 &
[3] 3609

```

```

[rrManatov@fedora ~]$ ./sem.sh 2 5 Выполнение > /dev/pts/2 &
[1] 5103
[rrManatov@fedora ~]$ ./sem.sh 3 4 Выход > /dev/pts/2 &
[2] 5221
[2]+  Завершён      ./sem.sh 3 4 Выход > /dev/pts/2
[rrManatov@fedora ~]$ ./sem.sh 3 5 Выполнени > /dev/pts/3 &
[2] 5678
Следующее действие:

```

Рисунок 3 Проверка работы кода

2) . Реализуем команду man с помощью командного файла. Изучите 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. (рис4) (рис5)

```
[rrManatov@fedora ~]$ cd /usr
[rrManatov@fedora usr]$ ls
bin  games  include  lib  lib64  libexec  local  sbin  share  src  tmp
[rrManatov@fedora usr]$ cd share/man/man1
[rrManatov@fedora man1]$ ls
.:1.gz
'[:1.gz'
ab.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-core.1.gz
abrt-action-analyze-java.1.gz
abrt-action-analyze-oops.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-oops-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-install-debuginfo.1.gz
abrt-action-list-dsos.1.gz
abrt-action-notify.1.gz
abrt-action-perform-ccpp-analysis.1.gz
abrt-action-save-package-data.1.gz
abrt-action-trim-files.1.gz
abrt-applet.1.gz
abrt-auto-reporting.1.gz
abrt-bodhi.1.gz
abrt-cli.1.gz
abrt-dump-journal-core.1.gz
abrt-dump-journal-oops.1.gz
abrt-dump-journal-xorg.1.gz
abrt-dump-oops.1.gz
abrt-dump-xorg.1.gz
abrt-handle-upload.1.gz
abrt-harvest-pstoreoops.1.gz
abrt-harvest-vmcore.1.gz
abrt-merge-pstoreoops.1.gz
abrt-retrace-client.1.gz
abrt-server.1.gz
```

Рисунок 4

Для данной задачи я создал файл man.h и написал код

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

Рисунок 5

Далее я проверил работу написанного скрипта и предварительно добавил право на выполнение файла (рис6) (рис7) (рис8)

```
[rrManatov@fedora ~]$ chmod +x man.sh
[rrManatov@fedora ~]$ ./man.sh ls
[rrManatov@fedora ~]$ ./man.sh mkdir
```

Рисунок 6

```
.\\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH LS "1" "July 2021" "GNU coreutils 8.32" "User Commands"
.SH NAME
ls \- list directory contents
.SH SYNOPSIS
.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=SIZE
with \fB\--l\fR, scale sizes by SIZE when printing them;
e.g., '\fB\--block-size=M\fR'; see SIZE format below
.TP
\fB\--B\fR, \fB\--ignore-backups\fR
do not list implied entries ending with ~
.TP
\fB\--c\fR
with \fB\--lt\fR: sort by, and show, ctime (time of last
modification of file status information);
with \fB\--l\fR: show ctime and sort by name;
otherwise: sort by ctime, newest first
.TP
\fB\--C\fR
list entries by columns
.TP
:
```

Рисунок 7

```

.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH MKDIR "1" "July 2021" "GNU coreutils 8.32" "User Commands"
.SH NAME
mkdir \- make directories
.SH SYNOPSIS
.B mkdir
[\fI[\fOPTION\fR]\fR... \fI[\fDIRECTORY\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[\fOm\fR, \fB[\fOB\fR[\fO-mode\fR=\fI[\fMODE\fR]\fR
set file mode (as in chmod), not a=rwx \- umask
.TP
\fB[\fOp\fR, \fB[\fOB\fR[\fO-parents\fR
no error if existing, make parent directories as needed
.TP
\fB[\fOv\fR, \fB[\fOB\fR[\fO-verbose\fR
print a message for each created directory
.TP
\fB[\fOZ\fR
set SELinux security context of each created directory
to the default type
.TP
\fB[\fO-context\fR[=\fI[\fCTX\fR]\fR
like \fB[\fOZ\fR, or if CTX is specified then set the SELinux
or SMACK security context to CTX
.TP
\fB[\fO-help\fR
display this help and exit
.TP
\fB[\fO-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 COPYRIGHT
:

```

Рисунок 8

3) Используя встроенную переменную \$RANDOM, напомним командный файл, генерирующий случайную последовательность букв латинского алфавита.

Для данной записи я создал файл random.sh и написал код рис(9)

```

#!/bin/bash
k=$1
for (( i = 0; i<$k; i++ ))
do
    ((char=$RANDOM%26+! ))
    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

```

Рисунок 9 код программы

```
[rrManatov@fedora ~]$ ./random.sh 7
yxafakh
[rrManatov@fedora ~]$ ./random.sh 15
[rrManatov@fedora ~]$ ./random.sh 15
yskzsoxysnxidh
```

Рисунок 10 код работает корректно

Вывод: : 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.

Контрольные вопросы:

1) while [$\$1 \neq \text{"exit"}$] 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 [" $\$1$ " $\neq \text{"exit"}$]

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 чисел от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT.

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 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