

Отчет к №12 лабораторной работе

Операционные системы

Газизова Регина

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	15
5	Контрольные вопросы	16

Список иллюстраций

3.1	Создание файла для 1 задания	7
3.2	Скрипт для 1 задания	8
3.3	Выполнение командного файла	8
3.4	Измененный скрипт (часть 1)	9
3.5	Измененный скрипт (часть 2)	10
3.6	Выполнение командного файла	11
3.7	Содержимое каталога	11
3.8	Скрипт для 2 задания	12
3.9	Выполнение командного файла	12
3.10	mkdir	12
3.11	rm	13
3.12	cat	13
3.13	Скрипт для 3 задания	13
3.14	Выполнение командного файла	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.

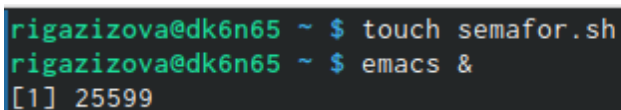
2 Задание

1. Написать 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). Запустить командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (`> /dev/tty#`, где `#` — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не в фоновом, а в привилегированном режиме. Доработать программу так, чтобы имелась возможность взаимодействия трёх или более процессов.
2. Реализовать команду `man` с помощью командного файла. Изучите содержимое каталога `/usr/share/man/man1`. В нём находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой `less` сразу же после просмотра содержимого 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`.
3. Используя встроенную переменную `RANDOM`, , — ., `RANDOM` выдаёт псевдослучайные числа в диапазоне от 0 до 32767

3 Выполнение лабораторной работы

1. Напишем 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 выполнения данной задачи создадим файл `semafor.sh` и откроем его в `emacs` (рис.1).



```
rigazizova@dk6n65 ~ $ touch semafor.sh
rigazizova@dk6n65 ~ $ emacs &
[1] 25599
```

Рис. 3.1: Создание файла для 1 задания

В файле напомним соответствующий скрипт (рис.2) и проверим его работу (команда `./semafor.sh 2 4`), предварительно добавив права на выполнение (команда `chmod +x semafor.sh`) (рис.3).

```
File Edit Options Buffers Tools Sh-Script Help
#!/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
```

U:*** semafor.sh All L23 (Shell)

Рис. 3.2: Скрипт для 1 задания

```
rigazizova@dk6n65 ~ $ chmod +x semafor.sh
rigazizova@dk6n65 ~ $ ./semafor.sh 2 4
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
```

Рис. 3.3: Выполнение командного файла

Затем изменим скрипт так, чтобы можно было запускать командный файл в

одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (рис.4-5).

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

Рис. 3.4: Измененный скрипт (часть 1)

```

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

```

Рис. 3.5: Измененный скрипт (часть 2)

Проверим его работу (например, команда `./semafor.sh 2 4 Ожидание > /dev/pts/1`) и увидим, что нам отказано в доступе. Но при этом скрипт работает корректно (рис.7) при вводе команды `./semafor.sh 2 4 Ожидание`.

```

rigazizova@edk6n65 ~ $ ./semafor.sh 2 4 Ожидание
Ожидание
^[[АОжидание
Следующее действие:

Следующее действие:

Следующее действие:

Следующее действие:

Следующее действие:

Следующее действие:

Следующее действие:

Следующее действие:

Следующее действие:
Выполнение
Выполнение
Выполнение
Выполнение
Следующее действие:
Выход
Выход

```

Рис. 3.6: Выполнение командного файла

2. Перед тем как приступить к выполнению 2 задания, изучим содержимое каталога `/usr/share/man/man1` (рис.8). В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд.

```

hugin_stacker.1.bz2      pipewire.1.bz2          zsh.1.bz2
hugin_stitch_project.1.bz2  ptop.1.bz2             zshbuiltins.1.bz2
humount.1.bz2           pk12util.1.bz2          zshcalsys.1.bz2
hunspell.1.bz2          pk2bzw.1.bz2            zshcompctl.1.bz2
hunzip.1.bz2            pkaction.1.bz2          zshcompsys.1.bz2
hvmgr.1.bz2             pkcheck.1.bz2           zshcompid.1.bz2
hvol.1.bz2              pkcon.1.bz2             zshcontrib.1.bz2
hwloc-annotate.1.bz2      pkexec.1.bz2            zshexon.1.bz2
hwloc-bind.1.bz2         pkg-config.1.bz2        zshmisc.1.bz2
hwloc-calc.1.bz2         pkgdata.1.bz2           zshmodules.1.bz2
hwloc-compress-dir.1.bz2  pkkill.1                zshoptions.1.bz2
hwloc-diff.1.bz2         pkmon.1.bz2             zshparam.1.bz2
hwloc-distrib.1.bz2      pktogf.1.bz2            zshroadmap.1.bz2
hwloc-dump-hwdata.1.bz2  pktopm.1.bz2            zshrcpsys.1.bz2
hwloc-gather-cpuid.1.bz2  pkttyagent.1.bz2        zshsoftsys.1.bz2
hwloc-gather-topology.1.bz2  pktty.1.bz2            zshzle.1.bz2
hwloc-info.1.bz2         pl2pm.1.bz2             zsoelim.1.bz2
hwloc-ls.1.bz2           planarg.1.bz2           zstd.1.bz2
hwloc-patch.1.bz2        planarity.1.bz2         zstdcat.1.bz2
hwloc-ps.1.bz2           plasmaengineexplorer.1.bz2  zstdgrep.1.bz2
hzip.1.bz2               plasmapkg2.1.bz2        zstdless.1.bz2
i3.1.bz2                 plasmafirewall.1.bz2    zvi-chains.1.bz2
i3bar.1.bz2              platex-dev.1            zvid.1.bz2
i3-config-wizard.1.bz2    play.1.bz2              zvi-ntsc-cc.1.bz2
i3-dmenu-desktop.1.bz2   plaympeg.1.bz2

```

Рис. 3.7: Содержимое каталога

Реализуем команду `man` с помощью командного файла. Для этого создадим файл `man.sh` и откроем его в `emacs`. Напишем скрипт для выполнения задания

(рис.9).

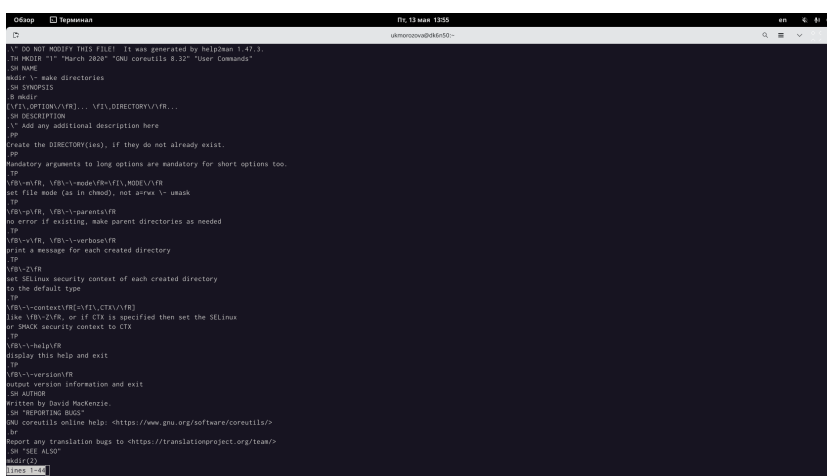
```
#!/bin/bash
a=$1
if [ -f /usr/share/man/man1/$a.1.bz2 ]
then
    bunzip2 -c /usr/share/man/man1/$1.1.bz2 | less
else
    echo "Справки по данной команде нет"
fi
```

Рис. 3.8: Скрипт для 2 задания

Проверим его работу (команды ./man.sh mkdir, ./man.sh rm, ./man.sh cat), предварительно дав ему право на выполнение с помощью команды chmod +x man.sh (рис.10). Результаты работы трех команд представлены на рисунках 11-13.

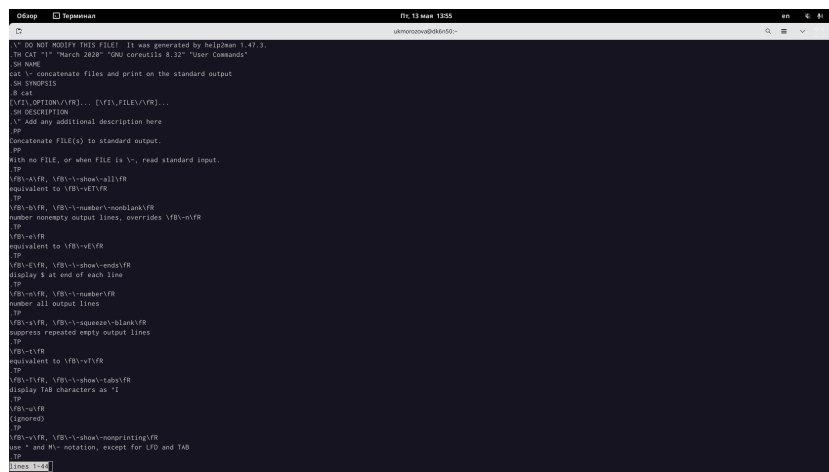
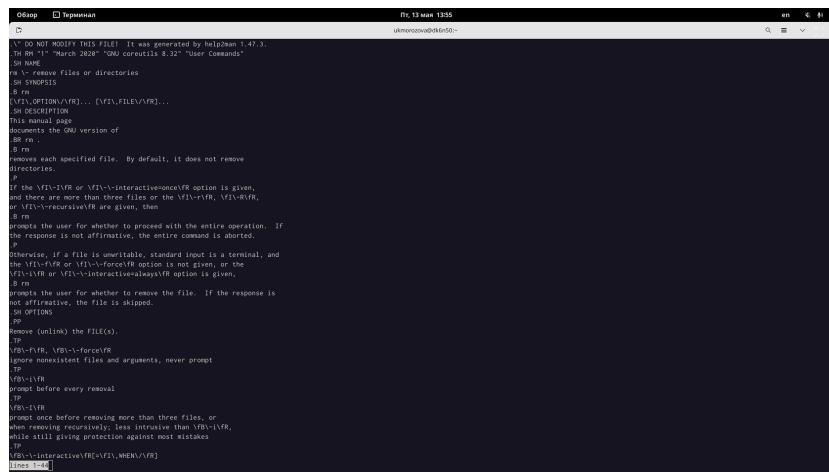
```
rigalizova@dk6n65 ~ $ chmod +x man.sh
rigalizova@dk6n65 ~ $ ./man.sh mkdir
rigalizova@dk6n65 ~ $ ./man.sh rm
rigalizova@dk6n65 ~ $ ./man.sh cat
```

Рис. 3.9: Выполнение командного файла



```
Обзор Терминал ftv:13 мая 18:55
AL! DO NOT MODIFY THIS FILE! It was generated by Help2man 1.41.2.
TH NAME "1" "March 2020" "GNU coreutils 8.32" "User Commands"
SH NAME
mkdir - make directories
SH SYNOPSIS
B mkdir
[V1,OPTION[V1R]]... V1, DIRECTORY[V1R]...
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
V1B-V1R, V1B-V1R-mode[V1R],MODE[V1R]
set file mode (as in chmod), not a-w-x or u-w-a-x
TP
V1B-V1R, V1B-V1R-parent[V1R]
no error if existing, make parent directories as needed
TP
V1B-V1R, V1B-V1R-verbose[V1R]
print a message for each created directory.
TP
V1B-V1R
set SELinux security context of each created directory
to the default type
TP
V1B-V1R-context[V1R],CTX[V1R]
like V1B-V1R, or if CTX is specified then set the SELinux
or SMACK security context to CTX
TP
V1B-V1R-help[V1R]
display this help and exit
TP
V1B-V1R-version[V1R]
output version information and exit
SH AUTHOR
Written by David MacKenzie.
SH "REPORTING BUGS"
GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
or
Report any translation bugs to <https://translationproject.org/team/>
SH SEE ALSO
mkdir(2)
lines 1-48
```

Рис. 3.10: mkdir



3. Используя встроенную переменную \$RANDOM, напомним командный файл, генерирующий случайную последовательность букв латинского алфавита. Для этого создадим файл random.sh и откроем его в emacs.

Напишем скрипт для выполнения 3 задания (рис.14).

```
#!/bin/bash
a=$1
for ((i=0; i<$a; 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
```

Проверим его работу (команда `./random.sh 158`), предварительно дав ему право на выполнение с помощью команды `chmod +x random.sh` (рис.15).

```
rigazizova@dk6n65 ~ $ chmod +x random.sh
rigazizova@dk6n65 ~ $ ./random.sh 120
jvvzouvmvftdrqbvgmekhujtdamtdstbqhctelholzabxkqtdsjyavxrgzyksgatuvcohwfmxbkfobnzmmvrmrosateiggyolycnggykfaxuhsvdcfu
rigazizova@dk6n65 ~ $
```

Рис. 3.14: Выполнение командного файла

4 Выводы

Я изучила основы программирования в оболочке ОС UNIX и научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 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="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 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` 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.