

# **Отчет по лабораторной работе №12**

**Операционные системы**

Морозова Ульяна Константиновна

# Список иллюстраций

0.1. Создание файла для 1 задания . . . . .	4
0.2. Скрипт для 1 задания . . . . .	5
0.3. Выполнение командного файла . . . . .	5
0.4. Измененный скрипт (часть 1) . . . . .	6
0.5. Измененный скрипт (часть 2) . . . . .	7
0.6. Выполнение командного файла . . . . .	7
0.7. Выполнение командного файла . . . . .	8
0.8. Содержимое каталога . . . . .	8
0.9. Скрипт для 2 задания . . . . .	9
0.10. Выполнение командного файла . . . . .	9
0.11. mkdir . . . . .	9
0.12. rm . . . . .	10
0.13. cat . . . . .	10
0.14. Скрипт для 3 задания . . . . .	11
0.15. Выполнение командного файла . . . . .	11

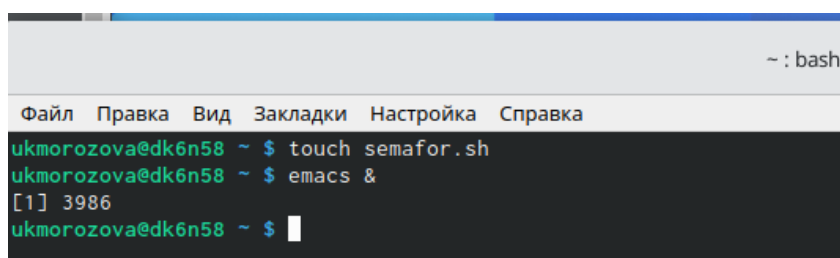
# Цель работы

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

# 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).



```
~ : bash
Файл  Правка  Вид  Закладки  Настройка  Справка
ukmorozova@dk6n58 ~ $ touch semafor.sh
ukmorozova@dk6n58 ~ $ emacs &
[1] 3986
ukmorozova@dk6n58 ~ $
```

Рис. 0.1.: Создание файла для 1 задания

В файле напишем 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
done

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

Рис. 0.2.: Скрипт для 1 задания

```
ukmorofova@dk6n58 ~ $ chmod +x semafor.sh
ukmorofova@dk6n58 ~ $ ./semafor.sh 2 4
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
```

Рис. 0.3.: 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. 0.4.: 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. 0.5.: Измененный скрипт (часть 2)

Проверим его работу (например, команда `./semafor.sh 2 4 Ожидание > /dev/pts/1`) и увидим, что нам отказано в доступе (рис.6). Но при этом скрипт работает корректно (рис.7) при вводе команды `./semafor.sh 2 4 Ожидание`.

```

ukmorofova@dk6n58 ~ $ ./semafor.sh 2 4 Ожидание > /dev/pts/1 &
[2] 5963
ukmorofova@dk6n58 ~ $ bash: /dev/pts/1: Отказано в доступе

[2]+  Выход 1          ./semafor.sh 2 4 Ожидание > /dev/pts/1
ukmorofova@dk6n58 ~ $ ./semafor.sh 2 4 Ожидание > /dev/pts/2 &
[2] 6139
ukmorofova@dk6n58 ~ $ bash: /dev/pts/2: Отказано в доступе

[2]+  Выход 1          ./semafor.sh 2 4 Ожидание > /dev/pts/2
ukmorofova@dk6n58 ~ $ ./semafor.sh 2 4 Выполнение > /dev/pts/1 &
[2] 6143
ukmorofova@dk6n58 ~ $ bash: /dev/pts/1: Отказано в доступе

[2]+  Выход 1          ./semafor.sh 2 4 Выполнение > /dev/pts/1
ukmorofova@dk6n58 ~ $ ./semafor.sh 2 4 Выполнение > /dev/pts/2 &
[2] 6167
ukmorofova@dk6n58 ~ $ bash: /dev/pts/2: Отказано в доступе

[2]+  Выход 1          ./semafor.sh 2 4 Выполнение > /dev/pts/2
ukmorofova@dk6n58 ~ $ █

```

Рис. 0.6.: Выполнение командного файла

```

ukmorofova@dk6n58 ~ $ ./semafor.sh 2 4 Ожидание
Ожидание
Ожидание
Следующее действие:
Ожидание
Ожидание
Ожидание
Следующее действие:
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Следующее действие:
Следующее действие:
Выход
Следующее действие:
Выход

```

Рис. 0.7.: 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.

```

htnotify.1.bz2      pic2graph.1.bz2    zipsplit.1.bz2
htop.1.bz2          pic2tpic.1x.bz2   ziptool.1.bz2
htpasswd.1.bz2      pickg.1.bz2       zless.1.bz2
htpurge.1.bz2       picom.1.bz2       zlib-decompress.1.bz2
htsearch.1.bz2      picom-trans.1.bz2 zlib-flate.1.bz2
htstat.1.bz2        piconv.1.bz2      zmore.1.bz2
httpcfg.1.bz2       piconv-3.10.0-perl-5.30.3.1.bz2 znew.1.bz2
httpt2dm.1.bz2      piconv-3.60.0-perl-5.32.1.1.bz2 zonetab2pot.py.1.bz2
hugin.1.bz2         picttopm.1.bz2    zresample.1.bz2
hugin_executor.1.bz2 pidgin.1.bz2       zretune.1.bz2
hugin_hdrmerge.1.bz2 pidof.1.bz2        zrun.1.bz2
hugin_lensdb.1.bz2  pinky.1.bz2        zsh.1.bz2
hugin_stacker.1.bz2 pipewire.1.bz2     zshall.1.bz2
hugin_stitch_project.1.bz2 pjtopm.1.bz2      zshbuiltins.1.bz2
humount.1.bz2       pk12util.1.bz2    zshcalsys.1.bz2
hunspell.1.bz2      pk2bm.1.bz2        zshcompctl.1.bz2
hunzip.1.bz2        pkaction.1.bz2     zshcompsys.1.bz2
hvmgr.1.bz2         pkcheck.1.bz2      zshcompwid.1.bz2
hvol.1.bz2          pkcon.1.bz2        zshcontrib.1.bz2
hwloc-annotate.1.bz2 pkexec.1.bz2       zshexpn.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 pkill.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 pktopbm.1.bz2     zshrcpsys.1.bz2
hwloc-gather-cpuuid.1.bz2 pkttyagent.1.bz2  zshzftpsys.1.bz2
hwloc-gather-topology.1.bz2 pkttype.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          plasmakg2.1.bz2    zstdless.1.bz2
i3.1.bz2            platex-dev.1       zvbi-chains.1.bz2
i3bar.1.bz2         play.1.bz2         zvbi.1.bz2
i3-config-wizard.1.bz2 plaympeg.1.bz2    zvbi-ntsc-cc.1.bz2
i3-dmenu-desktop.1.bz2
ukmorofova@dk6n58 /usr/share/man/man1 $

```

Рис. 0.8.: 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).

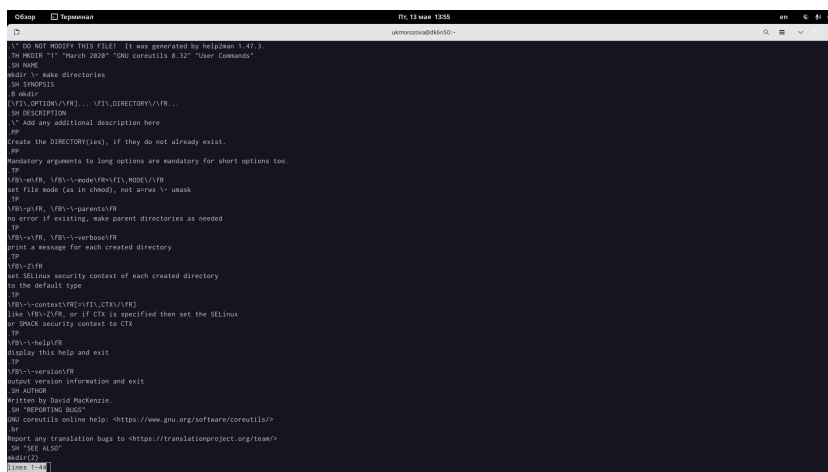
```
#!/bin/bash
a=$1
if [ -f /usr/share/man/man1/$a.1.bz2 ]
then
    bunzip2 -c /usr/share/man/man1/$1.1.bz2 | less
else
    echo "Справки по данной команде нет"
fi
```

Рис. 0.9.: 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.

```
ukmorofova@dk6n50 ~ $ ./man.sh mkdir
ukmorofova@dk6n50 ~ $ ./man.sh rm
ukmorofova@dk6n50 ~ $ ./man.sh cat
```

Рис. 0.10.: Выполнение командного файла



```

V: DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
M: Mkdir "1" "March 2020" "GNU coreutils 8.32" "User Commands"
M: NAME
M: mkdir - make directories
M: SYNOPSIS
M: mkdir
M: (V1, OPTION/VF)... (V1, DIRECTORY/VF)...
M: DESCRIPTION
M: V: Add any additional description here
M:
M: Create the DIRECTORY(ies), if they do not already exist.
M:
M: Mandatory arguments to long options are mandatory for short options too.
M:
M: VF/-m/VF, /VF/-mode/FF/FF, MODE/VF
M: set file mode (as in chmod), not -m/w/-u/wsk
M:
M: VF/-p/VF, /VF/-parents/VF
M: no error if existing, make parent directories as needed
M:
M: VF/-v/VF, /VF/-verbose/VF
M: print a message for each created directory
M:
M: VF/-Z/VF
M: set SELinux security context of each created directory
M: to the default type
M:
M: VF/--context/PFC/VF, CTX/VF
M: like VF/-Z/VF, or if CTX is specified then set the SELinux
M: or SMACK security context to CTX
M:
M: VF/-help/VF
M: Display this help and exit
M:
M: VF/-version/VF
M: Output version information and exit
M:
M:
M: Written by David Mackenzie.
M:
M: "REPORTING BUGS"
M:
M: GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
M:
M: Report any translation bugs to <https://translationproject.org/team/>
M:
M: "SEE ALSO"
M:
M: mkdir(2)
M:
M:

```

Рис. 0.11.: `mkdir`



```
#!/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
```

Рис. 0.14.: Скрипт для 3 задания

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

```
ukmorofova@dk6n50 ~ $ touch random.sh
ukmorofova@dk6n50 ~ $ chmod +x random.sh
ukmorofova@dk6n50 ~ $ ./random.sh 158
drpofsugdsqtmzuhguidvhcsnbhamlvhihjawxvprlptdocbnlwxadromssnvqqrzwxrcv1keolqmwetwqzgwugpdpptyybgrrnxlq
fntvcaidudxedrvtkufzxsutntsfuvxpssgmneiqidjxstboz
```

Рис. 0.15.: Выполнение командного файла

## Выводы

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

# 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.