

# **Лабораторная работа №12**

**Отчет по лабораторной работе**

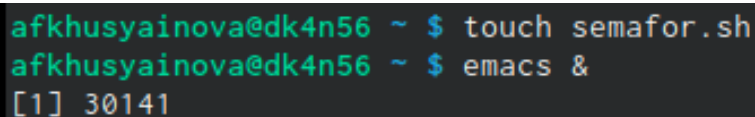
Хусяинова Адиля Фаритовна

# 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)



```
afkhusyainova@dk4n56 ~ $ touch semafor.sh
afkhusyainova@dk4n56 ~ $ emacs &
[1] 30141
```

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.2-3)

```
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 L1 (Shell-script[bash]) Чт мая 19 16:42 1.42
```

Рис. 0.2.: Скрипт для 1 задания

```
afkhusyainova@dk4n56 ~ $ chmod +x semafor.sh
afkhusyainova@dk4n56 ~ $ ./semafor.sh 1 2
Ожидание
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)

```

File Edit Options Buffers Tools Sh-Script Help
#!/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
}
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
        vypolnenie
    fi
    echo "Следующее действие: "
    read command
done
U:*** semafor.sh All L45 (Shell-script[bash]) Чт мая 19 16:45 1.02

```

Рис. 0.4.: Измененный скрипт

- 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 при вводе команды `./semafor.sh 2 4 Ожидание` (рис.5)

```

afkhusyainova@edk4n56 ~ $ ./semafor.sh 1 2 Ожидание > /dev/pts/1 &
[1] 32348
afkhusyainova@edk4n56 ~ $ bash: /dev/pts/1: Отказано в доступе

[1]+  Выход 1          ./semafor.sh 1 2 Ожидание > /dev/pts/1
afkhusyainova@edk4n56 ~ $ ./semafor.sh 1 2
Следующее действие:

Следующее действие:

Следующее действие:
Выход
Выход

```

Рис. 0.5.: 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 (рис.6)

```
hostname.1.bz2
hostnamectl.1.bz2
hpcdtopps.1.bz2
hpfddit.1.bz2
hpwd.1.bz2
hrename.1.bz2
hrmdir.1.bz2
hddm.1.bz2
hddig.1.bz2
hddigest.1.bz2
hddig-pdfparser.1.bz2
hddump.1.bz2
hdfuzzy.1.bz2
hload.1.bz2
hmerge.1.bz2
html2latex.1.bz2
htmldoc.1.bz2
HTMLinker.1.bz2
htmltree.1.bz2
hnotify.1.bz2
htop.1.bz2
htpasswd.1.bz2
htpurge.1.bz2
htsearch.1.bz2
htstat.1.bz2
httpcfg.1.bz2
htxt2dbm.1.bz2
hugin.1.bz2
hugin_executor.1.bz2
hugin_hdrmerge.1.bz2
hugin_lensdb.1.bz2
hugin_stacker.1.bz2
hugin_stitch_project.1.bz2
humount.1.bz2
humpall.1.bz2
hunzip.1.bz2
hvmgr.1.bz2
hvol.1.bz2
hwloc-annotate.1.bz2
hwloc-bind.1.bz2
hwloc-calc.1.bz2
hwloc-compress-dir.1.bz2
hwloc-diff.1.bz2
hwloc-distrib.1.bz2
hwloc-dump-hwdata.1.bz2
hwloc-gather-cpuid.1.bz2
hwloc-gather-topology.1.bz2
hwloc-info.1.bz2
hwloc-ls.1.bz2
hwloc-patch.1.bz2
hwloc-ps.1.bz2
hzip.1.bz2
i3.1.bz2
i3bar.1.bz2
i3-config-wizard.1.bz2
i3-dmnrdesktop.1.bz2
afkhusyainova@dk4n56 /usr/share/man/man1 $
pg_restore.1
pg_restore.1.bz2
pg_rewind.1.bz2
pg_standby.1.bz2
pg_standby13.1
pg_test_fsync.1
pg_test_fsync.1.bz2
pg_test_timing.1
pg_test_timing.1.bz2
pg_upgrade.1
pg_upgrade.1.bz2
pg_verifybackup.1
pg_verifybackup.1.bz2
pg_walddump.1
pg_walddump.1.bz2
pi1topps.1.bz2
pi1topbm.1.bz2
pic.1.bz2
pic2graph.1.bz2
pic2pic.1x.bz2
picke.1.bz2
picom.1.bz2
picom-trans.1.bz2
picov.1.bz2
picov-3.10.0-perl-5.30.3.1.bz2
picov-3.60.0-perl-5.32.1.1.bz2
pictopps.1.bz2
pidgin.1.bz2
pidof.1.bz2
pinky.1.bz2
pipewire.1.bz2
pjtopps.1.bz2
pk12util.1.bz2
pk2bm.1.bz2
pkaction.1.bz2
pkcheck.1.bz2
pkcon.1.bz2
pkexec.1.bz2
pkg-config.1.bz2
pkidata.1.bz2
pkill.1
pkmon.1.bz2
pktogf.1.bz2
pktopbm.1.bz2
pkttyagent.1.bz2
pktype.1.bz2
pl2pw.1.bz2
planarg.1.bz2
planarity.1.bz2
plasmaengineexplorer.1.bz2
plasmapk2.1.bz2
plasmoidviewer.1.bz2
latex-dev.1
play.1.bz2
plaympeg.1.bz2
yuvycsnoise.1.bz2
yuy2topps.1.bz2
zcat.1
zcmp.1
zdiff.1.bz2
ze1astopps.1.bz2
zenity.1.bz2
zforce.1.bz2
zgrep.1.bz2
zip.1.bz2
zipcloak.1.bz2
zipcmp.1.bz2
zipdetails.1.bz2
zipdetails-2.84.0-perl-5.30.3.1.bz2
zipdetails-2.93.0-perl-5.32.1.1.bz2
zipgrep.1.bz2
zipinfo.1.bz2
zipmerge.1.bz2
zipnote.1.bz2
zipsplit.1.bz2
ziptool.1.bz2
zless.1.bz2
zlib-decompress.1.bz2
zlib-flate.1.bz2
zmore.1.bz2
znew.1.bz2
zonetab2pot.py.1.bz2
zresample.1.bz2
zrestone.1.bz2
zrun.1.bz2
zsh.1.bz2
zshall.1.bz2
zshbuiltins.1.bz2
zshcalsys.1.bz2
zshcompctl.1.bz2
zshcompys.1.bz2
zshcompwid.1.bz2
zshcontrib.1.bz2
zshexpn.1.bz2
zshmisc.1.bz2
zshmodules.1.bz2
zshoptions.1.bz2
zshparam.1.bz2
zshroadmap.1.bz2
zshrcpsys.1.bz2
zshftpsys.1.bz2
zshzle.1.bz2
zsoella.1.bz2
zstd.1.bz2
zstdcat.1.bz2
zstdgrep.1.bz2
zstdless.1.bz2
zvbi-chains.1.bz2
zvbid.1.bz2
zvbi-ntsc-cc.1.bz2
```

Рис. 0.6.: Содержание 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.7)







```

#!/bin/dash
n=$((RANDOM%26+1))
for ((i=0; i<$n; 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. 0.12.: 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.14)

```

afkhusyainova@dk4n56 ~ $ chmod +x random.sh
[2]+  Завершён      emacs
afkhusyainova@dk4n56 ~ $ ./random.sh 123
muydcrmhfjlamzsnbbhpnaabgmdhbfvvlknghljohrtufaxnintpgqwsmyloojnkxsrdkvqkyvrjhhonsejctgwsiorwummeszjbhsosqgyxczdzlqxxjmv

```

Рис. 0.13.: 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="VAR2" echo "VAR3" : *Hello, World* • : 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 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 выполнять 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.