

# Лабораторная работа №14

Операционные системы

---

Павлова Т. Ю.

Российский университет дружбы народов, Москва, Россия

Целью данной лабораторной работы, является изучение основ программирования в оболочке ОС UNIX, а также научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.


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 и более процессов.
2. Реализовать 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

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: – оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; – C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд; – оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; – BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation). POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных

## Выполнение лабораторной работы

---

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 фоновом, а в привилегированном режиме. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов (рис. 1), (рис. 2).



```
#!/bin/bash

lockfile="./lock.file"
exec {fn}>$lockfile

while test -f "$lockfile"
do
if flock -n ${fn}
then
    echo "File is_blocked"
    sleep 5
    echo "File is unlocked"
    flock -u ${fn}
else
    echo "File is blocked"
    sleep 5
fi
done
```

```
[tanya@tatyana-pavlova ~]$ cd files_for_lab14
[tanya@tatyana-pavlova files_for_lab14]$ bash lab14-1.sh
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
```

Рис. 2: Компиляция файла

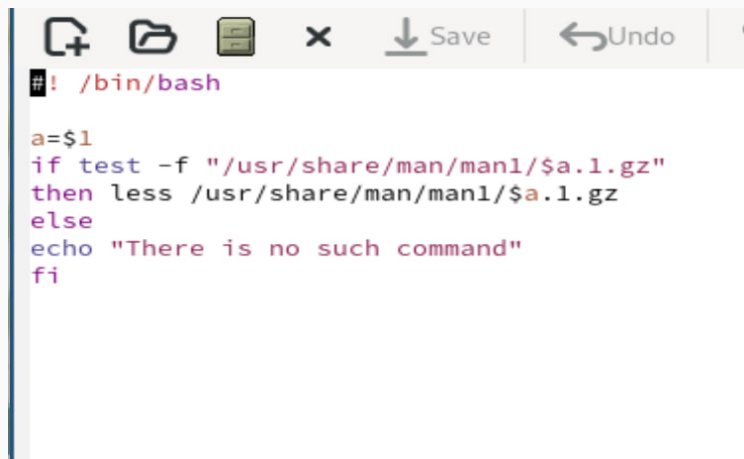


Реализовать команду `man` с помощью командного файла. Изучите содержимое каталога `/usr/share/man/man1`. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой `less` сразу же просмотрев содержимое справки. 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` (рис. 3), (рис. 4), (рис. 5), (рис. 6).

## Содержимое /usr/share/man/man1

```
[tanya@tatyana-pavlova ~]$ ls /usr/share/man/man1
.:1.gz
[.1.gz'
7z.1.gz
a2ping.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-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-list-dsos.1.gz
abrt-action-notify.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-server.1.gz
abrt-watch-log.1.gz
msgmerge.1.gz
msgunfmt.1.gz
msguniq.1.gz
mshortname.1.gz
mshowfat.1.gz
msxlint.1.gz
mttools.1.gz
mttoolstest.1.gz
mtrace.1.gz
mtx-babel.1.gz
mtx-base.1.gz
mtx-bibtex.1.gz
mtx-cache.1.gz
mtx-chars.1.gz
mtx-check.1.gz
mtx-colors.1.gz
mtx-context.1.gz
mtx-dvi.1.gz
mtx-epub.1.gz
mtx-evohome.1.gz
mtx-fcd.1.gz
mtx-flac.1.gz
mtx-fonts.1.gz
mtx-grep.1.gz
mtx-interface.1.gz
mtx-metapost.1.gz
mtx-modules.1.gz
mtx-package.1.gz
mtx-patterns.1.gz
mtx-pdf.1.gz
mtx-plain.1.gz
mtx-profile.1.gz
mtx-rsync.1.gz
mtxrun.1.gz
mtx-scite.1.gz
mtx-server.1.gz
```

Рис. 3: ls /usr/share/man/man1

A screenshot of a text editor window. The title bar at the top shows the file name 'Файл lab14-2.sh'. The editor's toolbar includes icons for opening, saving, and undoing, along with 'Save' and 'Undo' text labels. The code content is as follows:

```
#!/bin/bash

a=$1
if test -f "/usr/share/man/man1/$a.1.gz"
then less /usr/share/man/man1/$a.1.gz
else
echo "There is no such command"
fi
```

Рис. 4: lab14-2.sh

# Реализация команды man kill

```
ESC[4mKILLESC[24m(1)                                     User Commands
ESC[4mKILLESC[24m(1)

ESC[1mNAMEESC[0m
    kill - terminate a process

ESC[1mSYNOPSISESC[0m
    ESC[1mkill ESC[22mESC[1m-ESC[4mESC[22msignalESC[24mESC[1m-s ESC[4mESC[22msignalESC[24mESC[1m-pESC[22mESC[1m-q ESC[4mESC[22mvalueESC[24mESC[1m-aESC[22mESC[1m--timeout ESC[4mESC[22mmillisecondsESC[24mESC[4msignalESC[24mESC[1m--ESC[22mESC[4mpidESC[24mESC[4mnameESC[24m...

    ESC[1mkill -l ESC[22mESC[4mnumberESC[24m | ESC[1m-ESC[0m

ESC[1mDESCRIPTIONESC[0m
    The command ESC[1mkill ESC[22msends the specified ESC[4msignalESC[24m to the specified processes or process groups.

    If no signal is specified, the ESC[1mTERM ESC[22msignal is sent. The default action for this signal is to terminate the process. This signal should be used in preference to the ESC[1mKILL ESC[22msignal (number 9), since a process may install a handler for the TERM signal in order to perform clean-up steps before terminating in an orderly fashion. If a process does not terminate after a ESC[1mTERMESC[0m signal has been sent, then the ESC[1mKILL ESC[22msignal may be used; be aware that the latter signal cannot be caught, and so does not give the target process the opportunity to perform any clean-up before terminating.

    Most modern shells have a builtin ESC[1mkill ESC[22mcommand, with a usage rather similar to that of the command described here. The ESC[1m--allESC[22m, ESC[1m--pidESC[22m, and ESC[1m--queue ESC[22moptions, and the possibility to specify processes by command name, are local extensions.

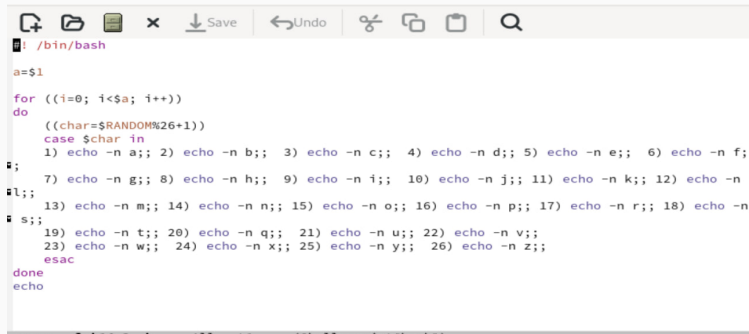
    If ESC[4msignalESC[24m is 0, then no actual signal is sent, but error checking is still performed.
/usr/share/man/man1/kill.1.gz
```

Рис. 5: Реализация команды man kill

```
[tanya@tatyana-pavlova ~]$ cd files_for_lab14  
[tanya@tatyana-pavlova files_for_lab14]$ chmod +x lab14-2.sh  
[tanya@tatyana-pavlova files_for_lab14]$ bash lab14-2.sh kill  
[tanya@tatyana-pavlova files_for_lab14]$
```

Рис. 6: Компиляция файла

Используя встроенную переменную `$RANDOM`, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что `$RANDOM` выдаёт псевдослучайные числа в диапазоне от 0 до 32767 (рис. 7), (рис. 8).

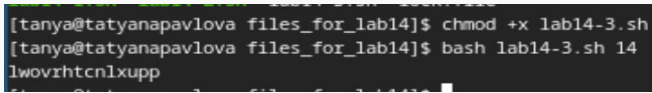


```
#!/bin/bash

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 r;; 18) echo -n
        s;; 19) echo -n t;; 20) echo -n q;; 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. 7: lab14-3.sh

A terminal window with a dark background and light-colored text. The prompt is [tanya@tatyana-pavlova files\_for\_lab14]. The first command is chmod +x lab14-3.sh. The second command is bash lab14-3.sh 14. The output of the script is lwovrhtcnlxupp.

```
[tanya@tatyana-pavlova files_for_lab14]$ chmod +x lab14-3.sh  
[tanya@tatyana-pavlova files_for_lab14]$ bash lab14-3.sh 14  
lwovrhtcnlxupp
```

Рис. 8: Компиляция файла



При выполнении данной лабораторной работы, я изучила основы программирования в оболочке ОС UNIX, а также научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.