

# Отчёт по лабораторной работе №12

## Дисциплина: операционные системы

Лебедева Алёна Алексеевна

### Содержание

Цель работы .....	1
Задание.....	1
Выполнение лабораторной работы .....	2
Контрольные вопросы.....	6
Выводы .....	8

### Цель работы

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

### 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 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 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.

- 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.

## Выполнение лабораторной работы

- Написала командный файл, реализующий упрощённый механизм семафоров. Командный файл в течение некоторого времени t1 дожидается освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени t2<>t1, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запускаю 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. Дорабатываю программу так, чтобы имела возможность взаимодействия трёх и более процессов. (рис. [-@fig:001])



```
1 #!/bin/bash
2 LOCKFILE="./lock.file"
3 exec {fn}>$LOCKFILE
4
5 if test -f "$LOCKFILE"
6 then
7     while
8         [ 1 = 1 ]
9     do
10         if flock -n ${fn}
11         then
12             echo "file block"
13             sleep 3
14             echo "file razblock"
15             flock -u ${fn}
16         else
17             echo "file block"
18             sleep 3
19         fi
20     done
21 fi
```

script01

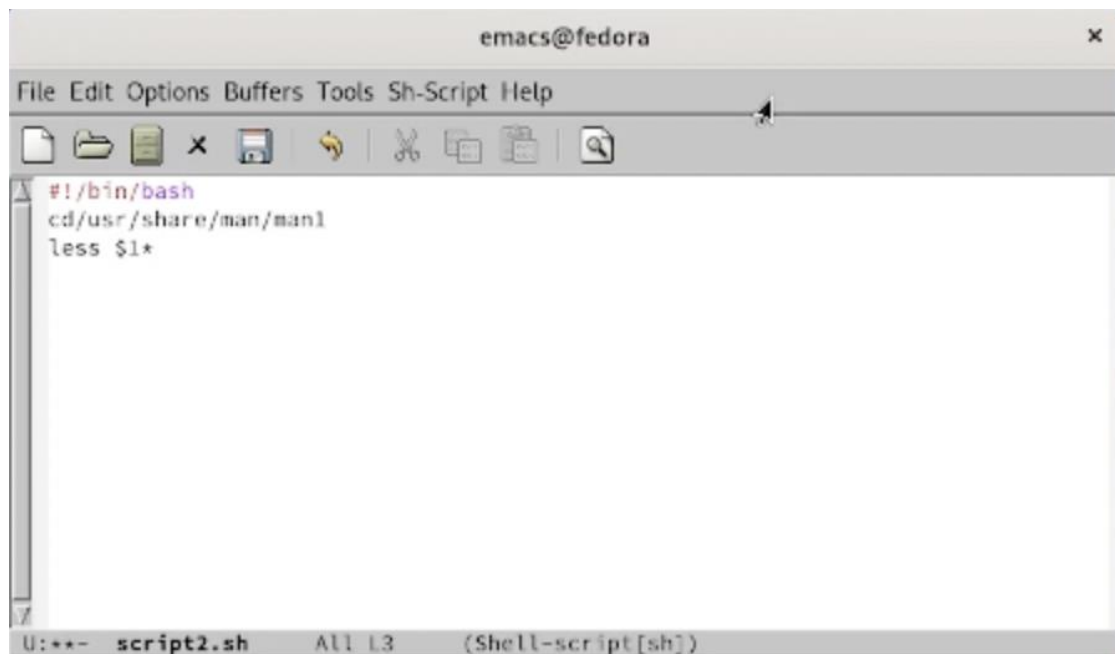
(рис. [-@fig:002])

```
[aalebedeva@fedora lab12]$ ./script1.sh
file block
file razblock
file block
file razblock
file block
^C
[aalebedeva@fedora lab12]$
```

*реализация*

2. Реализовываю команду man с помощью командного файла. Изучаю 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 получает 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.алога /tmp спомощью команды ls -lF, пояснила разницу между разными опциями

(рис. [-@fig:003])



*скринт 3*

(рис. [-@fig:004])

```

aalebedeva@fedora:~/work/study/2021-2022/Операционные с...
KILL(1) User Commands KILL(1)

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

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

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

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

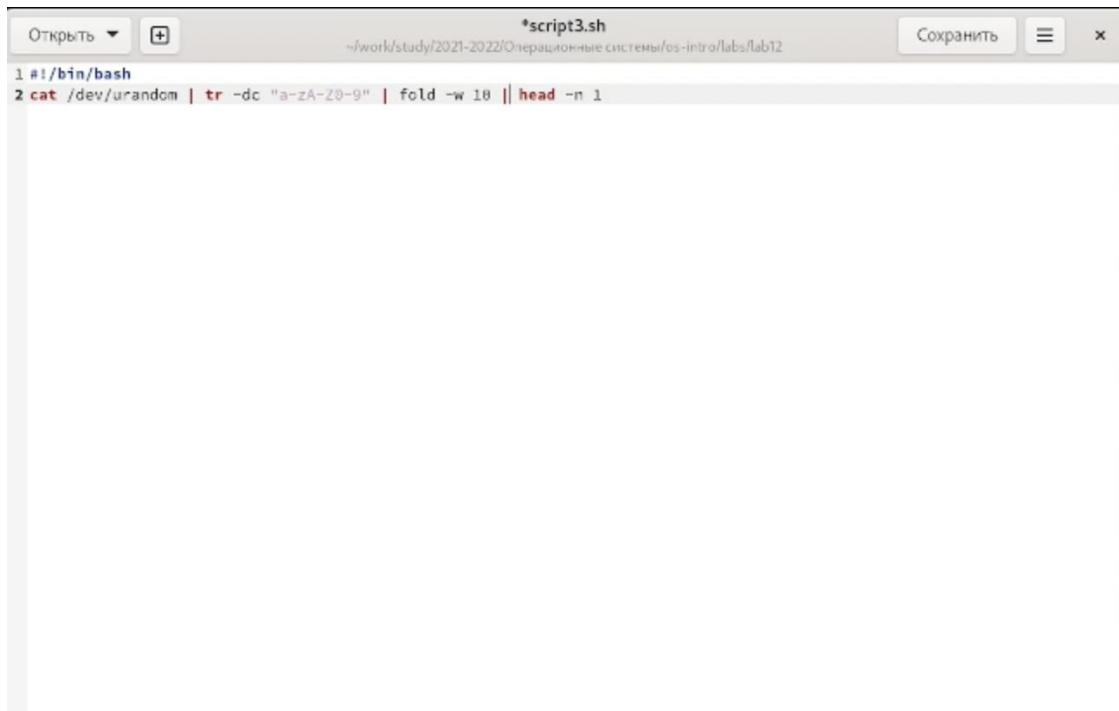
    If no signal is specified, the TERM signal is sent. The default action
for this signal is to terminate the process. This signal should be used
in preference to the KILL signal (number 9), since a process may
install a handler for the TERM signal in order to perform clean-up

kill.1.gz (file 1 of 2)
  
```

реализация

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

(рис. [-@fig:005])



The image shows a terminal window with a title bar containing the text `*script3.sh` and a file path `~/work/study/2021-2022/Операционные системы/os-intro/labs/lab12`. The window has buttons for "Открыть" (Open), "Сохранить" (Save), and a close button. The terminal content consists of two lines of commands:

```
1 #!/bin/bash
2 cat /dev/urandom | tr -dc "a-zA-Z0-9" | fold -w 10 || head -n 1
```

*скринт 3*

(рис. [-@fig:006])

```
aalebedeva@fedora:~/work/study/2021-2022/Операционные с...
PoL8GNLUGf80FLHH8CYBjvy6zqJthmUdeoYN77swJU5C5z1TRQ1MIFbYVWylc4fQy8rznR0PbDrzSHNf
8RPxkYSu5Dl8xEcEIF8Pez0WBo1KYZerQfPhayPtSZ1NXJhnP1xA0PAEyCLRew0LYdo7HgJkEPgbtFM
sFLWYSSBJAtAMNuRZscunf3XPJTjwRMTL7e0tyTYCpdWU72P3uFPzI2PDaMpZmv5C73QE4AQW79XgxR6
tFP5NRd88ueaFTRMgWRDBsIgUXEt5SNWZFasoJaQUQ5TrhZtEl21rwHz4JwiyVxNNQ15nZe6ZYXGyMy
1ECxbXqabY6PKolp0CZr4rzncXnP92AZTPrAtPB5fHEAsqDsmJ8L5eNU6gv0JraF0B0zY0QjTFeg8QIR
lK9HrzUUM7Arf4089FqJ38ub1glloj6B02SvVt5XlTx3roZHphjcbSd5YqH0QHvYqIX6A4Ncz0s1c3JA
hGhLP46u2t21FHZ37AA6glzeafA8fbbWR9fQ4gP2TAjQ8hMmfdyXfSnd1Eekxk30AVrcK6urdjFlrFG
Xc6V0FP1iHgIfKnHySsBxABW1Ak75LQFxxfabqeQKhK5CgmLedtPNJEKFXeVrQn1CLE9D5s1KpCVyZw
X5rRR6LY8IAwUVl1lDs73bp003kk89cZQEsbr3LiYLlU7jb2V04c408QglqVAA9wEG6tmCauq5NyGIyh
FBm4XrbWwGPlowmH176hZh4jZWePX3ajSxflRWVLF1kLT1kw68EsBZVroFU7I6Rme51W7r50vExnSheB
4qVYDGRVlMiX612XTBFkdGIl3eQx1A0jy3nKthwfwUZ92tLuKwIM0ZaWcG4GS10mWwY66ETjuOnVpreEQ
hA60FJy9Ny6RkJBX5Sr0gbpgg8PSCDHy6iIY76EBt0C19sB071t2gqgyGC8u71QAfktW2n3Fn1507sAc
iPugV2ML7KFw2ziMjeJCi6lxOB8NtecBo0CzCmefs4mISDPnIF0IWH0h1BfLNMt7AvKxCPPxIHZ4x6FM
MBR0Qk6iQqAqPjW36u46DciLMEjNAakyiSchBchYmCtkfeuF5Yp1f3LdhA5sAl2yDBn1Bm7WK1ZCIg7k
eXcYHQSR6wgfown0Vz5Fel7tJlt4mmHJj0K8TRFZF380FzoApJrqibe0AeFaNxs1fwXGYAyjXt2Z4Lri
VIFSXYL8MMjIOqEH0ZT1B6pXrzt9Uh2Wn4zaY0TlrIzXw29cYJyJew1TMhnZP0EeQgzkkRSFLZTNkIsB
uugXaioesN5esaMuvUicZ9DW3vDGVAj^C
[aalebedeva@fedora lab12]$ ./script3.sh
VRqBk1Ajr3
[aalebedeva@fedora lab12]$ ./script3.sh
DcjnyTgTKF
[aalebedeva@fedora lab12]$ ./script3.sh
Cq9Gk7MK4Y
[aalebedeva@fedora lab12]$
```

реализация

## Контрольные вопросы

1. В строке while [ $\$1 \neq \text{"exit"}$ ] квадратные скобки надо заменить на круглые.
2. Есть несколько видов конкатенации строк. Например, VAR1="Hello," VAR2="World" VAR3="VAR1VAR2" echo "\$VAR3"
3. Команда seq выводит последовательность целых или действительных чисел, подходящую для передачи в другие программы. В bash можно использовать seq с циклом for, используя подстановку команд. Например, \$ for i in \$(seq 1 0.5 4) do echo "The number is \$i" done
4. Результатом вычисления выражения  $\$((10/3))$  будет число 3.
5. Список того, что можно получить, используя Z Shell вместо Bash: Встроенная команда zmv поможет массово переименовать файлы/директории, например, чтобы добавить '.txt' к имени каждого файла, запустите zmv -C '(\*)(#q.)' '\$1.txt'. Утилита zcalc — это замечательный калькулятор командной строки, удобный способ считать быстро, не покидая терминал. Команда zparseopts — это однострочник, который поможет разобрать сложные варианты, которые предоставляются скрипту. Команда autopushd позволяет делать popd после того, как с помощью cd, чтобы вернуться в предыдущую директорию.

Поддержка чисел с плавающей точкой (коей Bash не содержит). Поддержка для структур данных «хэш». Есть также ряд особенностей, которые присутствуют только в Bash: Опция командной строки `-porc`, которая позволяет пользователю иметь дело с инициализацией командной строки, не читая файл `.bashrc` Использование опции `-rcfile` с `bash` позволяет исполнять команды из определённого файла. Отличные возможности вызова (набор опций для командной строки) Может быть вызвана командой `sh` Bash можно запустить в определённом режиме POSIX. Примените `set -o posix`, чтобы включить режим, или `--posix` при запуске. Можно управлять видом командной строки в Bash. Настройка переменной `PROMPT_COMMAND` с одним или более специальными символами настроит её за вас. Bash также можно включить в режиме ограниченной оболочки (с `rbash` или `-restricted`), это означает, что некоторые команды/действия больше не будут доступны: Настройка и удаление значений служебных переменных `SHELL`, `PATH`, `ENV`, `BASH_ENV` Перенаправление вывода с использованием операторов `'>'`, `'>|'`, `'<>'`, `'>&'`, `'&>'`, `'>>'` Разбор значений `SHELLOPTS` из окружения оболочки при запуске Использование встроенного оператора `exes`, чтобы заменить оболочку другой командой

6. Синтаксис конструкции `for ((a=1; a <= LIMIT; a++))` верен.
7. Язык `bash` и другие языки программирования: -Скорость работы программ на ассемблере может быть более 50% медленнее, чем программ на `си/си++`, скомпилированных с максимальной оптимизацией; -Скорость работы виртуальной ява-машины с байт-кодом часто превосходит скорость аппаратуры с кодами, получаемыми трансляторами с языков высокого уровня. Ява-машина уступает по скорости только ассемблеру и лучшим оптимизирующим трансляторам; -Скорость компиляции и исполнения программ на яваскрипт в популярных браузерах лишь в 2-3 раза уступает лучшим трансляторам и превосходит даже некоторые качественные компиляторы, безусловно намного (более чем в 10 раз) обгоняя большинство трансляторов других языков сценариев и подобных им по скорости исполнения программ; -Скорость кодов, генерируемых компилятором языка `си` фирмы Intel, оказалась заметно меньшей, чем компилятора GNU и иногда LLVM; -Скорость ассемблерных кодов `x86-64` может меньше, чем аналогичных кодов `x86`, примерно на 10%; -Оптимизация кодов лучше работает на процессоре Intel; -Скорость исполнения на процессоре Intel была почти всегда выше, за исключением языков лисп, эрланг, аук (`gawk`, `mawk`) и бэш. Разница в скорости по бэш скорее всего вызвана разными настройками окружения на тестируемых системах, а не собственно транслятором или железом. Преимущество Intel особенно заметно на 32-разрядных кодах; -Стек большинства тестируемых языков, в частности, ява и яваскрипт, поддерживают только очень ограниченное число рекурсивных вызовов. Некоторые трансляторы (`gcc`, `icc`, ...) позволяют увеличить размер стека изменением переменных среды исполнения или параметром; -В рассматриваемых версиях `gawk`, `php`, `perl`, `bash` реализован динамический стек,

позволяющий использовать всю память компьютера. Но perl и, особенно, bash используют стек настолько экстенсивно, что 8-16 ГБ не хватает для расчета `ask(5,2,3)`

## Выводы

В ходе выполнения лабораторной работы я изучила основы программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.