

# **Лабораторная работа №12**

**Программирование в командном процессоре ОС UNIX. Расширенное  
программирование**

Ежова Алиса Михайловна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>4</b>	<b>Выводы</b>	<b>11</b>
<b>5</b>	<b>Контрольные вопросы</b>	<b>12</b>

## Список иллюстраций

3.1	Создание файла . . . . .	7
3.2	Написание кода . . . . .	8
3.3	Результат . . . . .	8
3.4	Создание файла . . . . .	9
3.5	Написание кода . . . . .	9
3.6	Результат . . . . .	9
3.7	Результат . . . . .	9
3.8	Создание файла . . . . .	9
3.9	Написание кода . . . . .	10
3.10	Результат . . . . .	10

# 1 Цель работы

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

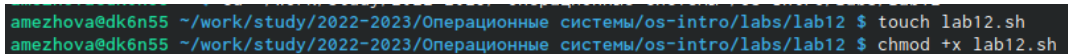
## 2 Задание

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 фоновом, а в привилегированном режиме. Доработать программу так, чтобы имела 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`.
3. 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.

### 3 Выполнение лабораторной работы

- 1) Написала 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, использовала его в течение некоторого времени  $t_2 < t_1$ , также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустила 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. Доработала программу так, чтобы имела возможность взаимодействия трёх и более процессов.

A screenshot of a terminal window showing two commands being executed. The first command is 'touch lab12.sh' and the second is 'chmod +x lab12.sh'. Both commands are executed from the directory ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12. The terminal output shows the commands being executed successfully.

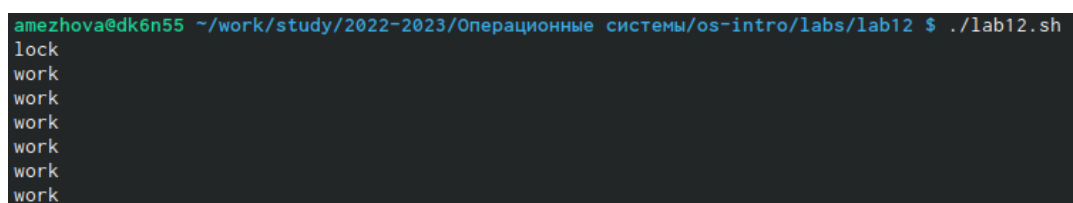
```
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ touch lab12.sh
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ chmod +x lab12.sh
```

Рис. 3.1: Создание файла



```
1 #!/bin/bash
2 lockfile="./lockfile"
3 exec {fn}>$lockfile
4 echo "lock"
5 until flock -n ${fn}
6 do
7     echo "not lock"
8     sleep 1
9     flock -n ${fn}
10 done
11 for ((i=0; i<=5; i++))
12 do
13     echo "work"
14     sleep 1
15 done
```

Рис. 3.2: Написание кода



```
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ ./lab12.sh
lock
work
work
work
work
work
work
```

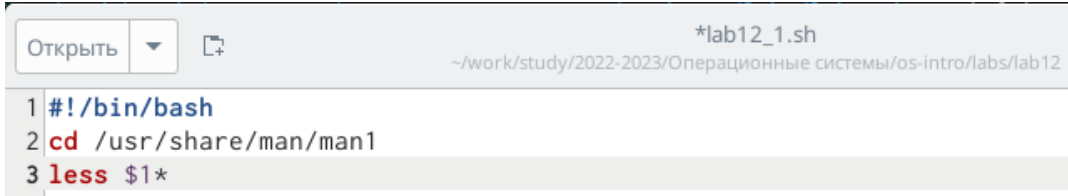
Рис. 3.3: Результат

- 2) Реализовала команду man с помощью командного файла. Изучила 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.



```
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ touch lab12_1.sh
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ chmod +x lab12_1.sh
```

Рис. 3.4: Создание файла

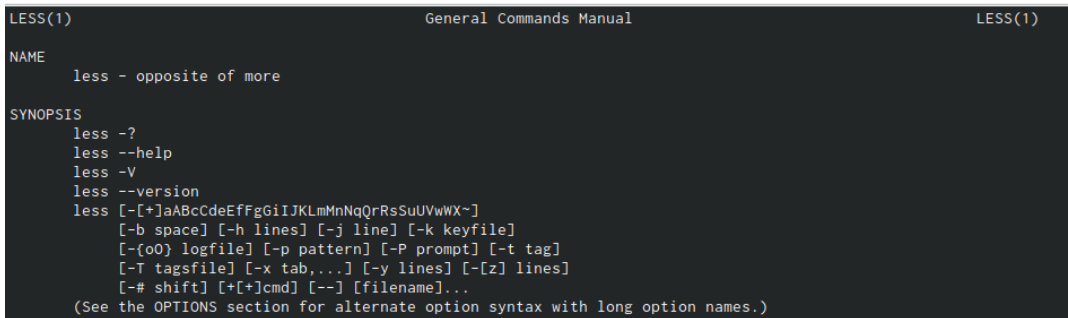


```
1 #!/bin/bash
2 cd /usr/share/man/man1
3 less $1*
```

Рис. 3.5: Написание кода

```
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ ./lab12_1.sh less
```

Рис. 3.6: Результат



```
LESS(1)                                     General Commands Manual                                     LESS(1)

NAME
    less - opposite of more

SYNOPSIS
    less -?
    less --help
    less -V
    less --version
    less [-[+]aABcCdeEffGgiIJKLmMnNqQrRsSuUVwWX~]
        [-b space] [-h lines] [-j line] [-k keyfile]
        [-{o0} logfile] [-p pattern] [-P prompt] [-t tag]
        [-T tagsfile] [-x tab,...] [-y lines] [-[z] lines]
        [-# shift] [+{+}cmd] [--] [filename]...
    (See the OPTIONS section for alternate option syntax with long option names.)
```

Рис. 3.7: Результат

- 3) Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

```
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ touch lab12_2.sh
amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ chmod +x lab12_2.sh
```

Рис. 3.8: Создание файла

A screenshot of a text editor window with a menu bar (File, Edit, Options, Buffers, Tools, Sh-Script, Outline, Hide/Show, Help) and a light yellow background. The script content is as follows:

```
#!/bin/bash
M=8
c=1
d=1
echo
echo "8 random words"
while ((c≠(($M+1))))
do
    echo $(for((i=1;i<8;i++)); do printf '$s' "${RANDOM:0:1}"; done) | tr '0-9' 'a-z'
    echo $d
    ((c+=1))
    ((d+=1))
done
```

Рис. 3.9: Написание кода

A screenshot of a terminal window with a dark background. The prompt is 'amezhova@dk6n55 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 \$'. The command './lab12\_2.sh' has been executed, resulting in the following output:

```
8 random words
$$$$$$$$$
1
$$$$$$$$$
2
$$$$$$$$$
3
$$$$$$$$$
4
$$$$$$$$$
5
$$$$$$$$$
6
$$$$$$$$$
7
$$$$$$$$$
8
```

Рис. 3.10: Результат

## 4 Выводы

В ходе выполнения Лабораторной работы №12, я изучила основы программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 5 Контрольные вопросы

1. В строке `while [$1 != "exit"]` квадратные скобки надо заменить на круглые.
2. Есть несколько видов конкатенации строк. Например, `VAR1="Hello," VAR2="World" VAR3="VAR1VAR2" echo "$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 ис-

ключением 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)