

Лабораторная работа №12

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

Бекназарова Виктория Тиграновна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	10
4	Контрольные вопросы	11

Список иллюстраций

2.1	Задание 1	7
2.2	Задание 2	8
2.3	Задание 2	8
2.4	Задание 3	9

Список таблиц


1 Цель работы

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

2 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). Запустим 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. 2.1).

```
vtbeknazarova@dk2n25 ~ $ cd ~/work/study/2022-2023/"Операционные системы"/os-intro/labs/lab12
vtbeknazarova@dk2n25 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ touch lab12.sh
vtbeknazarova@dk2n25 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ chmod +x lab12.sh
vtbeknazarova@dk2n25 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ ./lab12.sh
```



```
1 #!/bin/bash
2 lockfile="./lockfile"
3 exec {fn}>$lockfile
4 exec "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. 2.1: Задание 1

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` (рис. 2.2).(рис. 2.3).

```
LESS(1)

NAME
    less - opposite of more

SYNOPSIS
    less -?
    less --help
    less -V
    less --version
    less [-[+]aABcCdeEffGgiIJKLmMnNqQrRsSuUVvWwX~]
        [-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.)
```

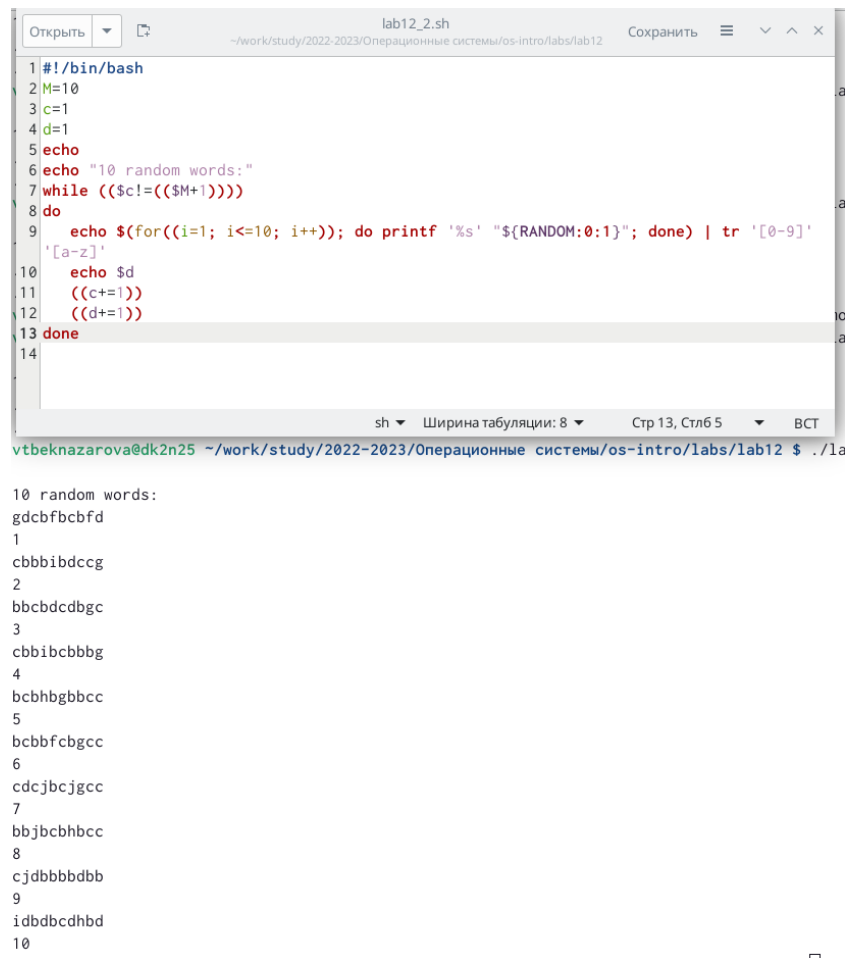
Рис. 2.2: Задание 2

```
рационные системы/os-intro/labs/lab12 $
рационные системы/os-intro/labs/lab12 $
рационные системы/os-intro/labs/lab12 $ chmod +x lab12_1.sh
рационные системы/os-intro/labs/lab12 $ ./lab12_1.sh
рационные системы/os-intro/labs/lab12 $ ./lab12_1.sh less
рационные системы/os-intro/labs/lab12 $
```

The screenshot shows a terminal window with a script being executed. The script's output is visible in a separate window titled 'lab12_1.sh', which shows the execution of 'cd /usr/share/man/man1' and 'less \$1*'. The terminal window also shows the command 'less' being entered to view the output of the script.

Рис. 2.3: Задание 2

- 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. 2.4).



The image shows a terminal window titled 'lab12_2.sh' with a file path of '~/.work/study/2022-2023/Операционные системы/os-intro/labs/lab12'. The script contains the following code:

```
1#!/bin/bash
2M=10
3c=1
4d=1
5echo
6echo "10 random words:"
7while (($c!=($M+1)))
8do
9    echo $(for((i=1; i<=10; i++)); do printf '%s' "${RANDOM:0:1}"; done | tr '[:0-9:]'
10    '[a-z]')
11    ((c+=1))
12    ((d+=1))
13done
14
```

The terminal output shows the execution of the script, resulting in 10 random words:

```
10 random words:
gdcfbfcbfd
1
cbbibdbccg
2
bbcbdcdbgc
3
cbbibcbbbg
4
bcbhgbbcc
5
bcbbfcbgcc
6
cdcjbcjgcc
7
bbjbcbhbcc
8
cjdbsbbdbb
9
idbdbcdhbd
10
```

Рис. 2.4: Задание 3

3 Выводы

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

4 Контрольные вопросы

1. В строке `while [$1 != "exit"]` квадратные скобки надо заменить на круглые.
2. Есть несколько видов конкатенации строк. Например, `VAR1="Hello," VAR2="World" VAR3="${VAR1}${VAR2}" echo "$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,¹⁷ 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 18 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)