

Лабораторная работа №12

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

Галиева Аделина Руслановна

Содержание

| | | |
|---|--------------------------------|----|
| 1 | Цель работы | 5 |
| 2 | Выполнение лабораторной работы | 6 |
| 3 | Контрольные вопросы | 9 |
| 4 | Выводы | 12 |
| | Список литературы | 13 |

Список иллюстраций

| | | |
|-----|---------------------|---|
| 2.1 | Задание 1 | 6 |
| 2.2 | Задание 2 | 7 |
| 2.3 | Задание 3 | 7 |
| 2.4 | Задание 4 | 8 |

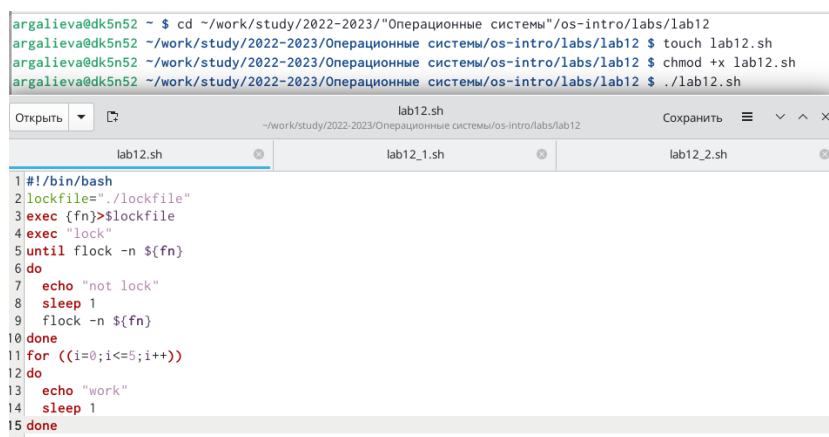
Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

1. Пишем командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 дожидаться 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). Запускаем 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. (рис. 2.1)



```
argalievadk5n52 ~ $ cd ~/work/study/2022-2023/"Операционные системы"/os-intro/labs/lab12
argalievadk5n52 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ touch lab12.sh
argalievadk5n52 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ chmod +x lab12.sh
argalievadk5n52 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ ./lab12.sh

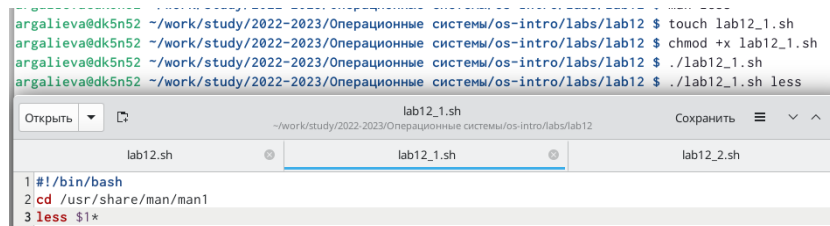
lab12.sh
1 #!/bin/bash
2 lockfile="./lockfile"
3 exec {fn}>$lockfile
4 exec "lock"
5 until flock -n ${fn}
6 do
7     echo "not lock"
8     sleep 1
9     flock -n ${fn}
10 done
11 for ((i=0;i<=5;i++))
12 do
13     echo "work"
14     sleep 1
15 done
```

Рис. 2.1: Задание 1

2. Реализуем 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)

```
argalieva@dk5n52 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ touch lab12_1.sh
argalieva@dk5n52 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ chmod +x lab12_1.sh
argalieva@dk5n52 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ ./lab12_1.sh
argalieva@dk5n52 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ ./lab12_1.sh less
```



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

Рис. 2.2: Задание 2

```
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 tagfile] [-x tab,...] [-y lines] [-[z] lines]
        [-# shift] [+{+}cmd] [--] [filename]...
    (See the OPTIONS section for alternate option syntax with long option names.)

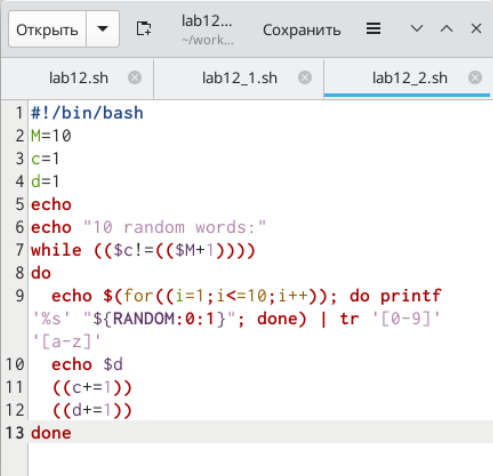
DESCRIPTION
```

Рис. 2.3: Задание 3

- Используем встроенную переменную \$RANDOM, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. (рис. 2.4)

```
argalievadk5n52 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12
$ touch lab12_2.sh
argalievadk5n52 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12
$ chmod +x lab12_2.sh
argalievadk5n52 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12
$ ./lab12_2.sh

10 random words:
gijccbccdc
1
cbgbbicdib
2
cbbbfccbgc
3
cjcccdcbdb
4
bcbbdcbcc
5
cbccbbbcc
6
bcccbefbcg
7
hbcbgbbfi
8
bbcbcedhcd
```



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

Рис. 2.4: Задание 4

3 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 10.5 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 (к которой `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 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)

4 Выводы

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

Список литературы