

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №12

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

Анастасия Павловна Баранова, НБИбд-01-21

Содержание

1	Цель работы	4
2	Задание	5
3	Выполнение лабораторной работы	7
4	Вывод	11
5	Ответы на контрольные вопросы	12

Список иллюстраций

3.1	Напишу командный файл.	7
3.2	Демонстрирую работу командного файла.	8
3.3	Напишу командный файл.	8
3.4	Демонстрирую работу командного файла.	8
3.5	Демонстрирую работу командного файла.	9
3.6	Напишу командный файл.	9
3.7	Демонстрирую работу командного файла.	10

1 Цель работы

Целью данной лабораторной работы является изучить основы программирования в оболочке ОС UNIX и 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 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`.
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

Напишу 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). Запущу 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. Доработаю программу так, чтобы имелась возможность взаимодействия трёх и более процессов. (рис. 3.1, рис. 3.2)



```
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. 3.1: Напишу командный файл.

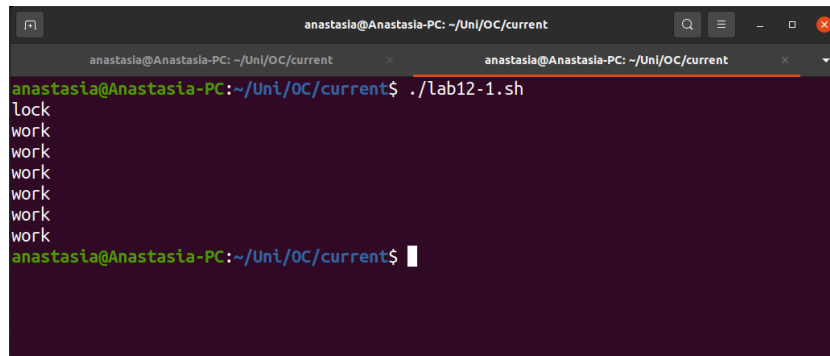
A terminal window titled 'anastasia@Anastasia-PC: ~/Uni/OC/current'. The prompt is 'anastasia@Anastasia-PC:~/Uni/OC/current\$'. The user enters './lab12-1.sh'. The script outputs 'lock', 'work', 'work', 'work', 'work', 'work', 'work', and 'work' on separate lines. The prompt returns to 'anastasia@Anastasia-PC:~/Uni/OC/current\$'.

Рис. 3.2: Демонстрирую работу командного файла.

Реализую команду `man` с помощью командного файла. Изучу 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. 3.3, рис. 3.4, рис. 3.5)

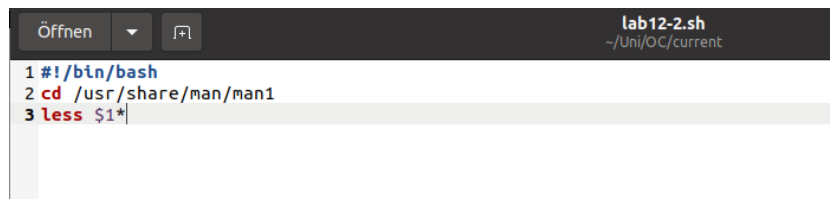
A code editor window titled 'lab12-2.sh' with the path '~/Uni/OC/current'. It shows three lines of code: '1 #!/bin/bash', '2 cd /usr/share/man/man1', and '3 less \$1*'. The first line is blue, the second is red, and the third is red.

Рис. 3.3: Напишу командный файл.

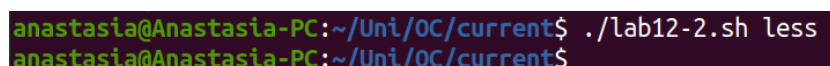
A terminal window showing the execution of the script. The prompt is 'anastasia@Anastasia-PC:~/Uni/OC/current\$'. The user enters './lab12-2.sh less'. The prompt returns to 'anastasia@Anastasia-PC:~/Uni/OC/current\$'.

Рис. 3.4: Демонстрирую работу командного файла.


```
anastasia@Anastasia-PC: ~/Uni/OC/current
.TH LESS 1 "Version 551: 11 Jun 2019"
.SH NAME
less \- opposite of more
.SH SYNOPSIS
.B "less \-?"
.br
.B "less \-.-help"
.br
.B "less \-V"
.br
.B "less \-.-version"
.br
.B "less [\+][a-zA-ZcdeFFgGIjKlMnNqRrSsUuVvWwX-]"
.br
.B "      [\+b \fIspace\| \fP] [\+h \fIlines\| \fP] [\+j \fIline\| \fP] [\+k \fIkeyfile\| \fP]"
.br
.B "      [\+o} \fIlogfile\| \fP] [\+p \fIpattern\| \fP] [\+P \fIprompt\| \fP] [\+t \fItag\| \fP]"
.br
.B "      [\+T \fItagsfile\| \fP] [\+x \fItab\| \fP,...] [\+y \fIlines\| \fP] [\+z \fIlines\| \fP]"
.br
.B "      [\+ # \fIshift\| \fP] [\+][\+]\fIcmd\| \fP] [\+ \-] [\fIfilename\| \fP]..."
.br
(See the OPTIONS section for alternate option syntax with long option names.)

.SH DESCRIPTION
.I Less
is a program similar to
.I more
(1), but it has many more features.
.I less
does not have to read the entire input file before starting,
so with large input files it starts up faster than text editors like
.I vi
```

Рис. 3.5: Демонстрирую работу командного файла.

Используя встроенную переменную \$RANDOM, напишу командный файл, генерирующий случайную последовательность букв латинского алфавита. Учту, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767. (рис. 3.6, рис. 3.7)

```
lab12-3.sh
~/Uni/OC/current
1 #!/bin/bash
2 M=10
3 c=1
4 d=1
5 echo
6 echo "10 random words: "
7 while ((S!=((M+i))))
8 do
9     echo $(for((i=1;i<=10;i++)); do printf '%s' "${RANDOM:0:1}"; done) | tr '0-9' '[a-z]'
10    echo $d
11    ((c+=1))
12    ((d+=1))
13 done
```

Рис. 3.6: Напишу командный файл.

```
anastasia@Anastasia-PC: ~/Uni/OC/current
anastasia@Anastasia-PC:~/Uni/OC/current$ touch lab12-3.sh
anastasia@Anastasia-PC:~/Uni/OC/current$ chmod +x lab12-3.sh
anastasia@Anastasia-PC:~/Uni/OC/current$ ./lab12-3.sh

10 random words:
bbdbccctlb
1
gbcbbccbgc
2
lbdgcbbcc
3
ecbcbdbcb
4
bdcdccbbcc
5
bccchbcjb
6
cdcbcheddd
7
ieddccgggc
8
ccbcgjcjcc
9
dcbbeggbhc
10
anastasia@Anastasia-PC:~/Uni/OC/current$
```

Рис. 3.7: Демонстрирую работу командного файла.

4 Вывод

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

5 Ответы на контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `while [$1 != "exit"]`

Ответ: В строке `while [$1 != "exit"]` квадратные скобки надо заменить на круглые.

2. Как объединить (конкатенация) несколько строк в одну? Ответ: Есть несколько видов конкатенации строк. Например, `VAR1="Hello," VAR2="World" VAR3="VAR1VAR2" echo "$VAR3"`

3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать её функционал при программировании на `bash`? Ответ: 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, `(seq 1 0.5 4) do echo "The number is $i" done`.

4. Какой результат даст вычисление выражения `$((10/3))`? Ответ: Результатом вычисления выражения `$((10/3))` будет число 3.

5. Укажите кратко основные отличия командной оболочки `zsh` от `bash`. Ответ: 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 `pwd` 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 `-norc`, 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 `exec`, чтобы заменить оболочку другой командой.

6. Проверьте, верен ли синтаксис данной конструкции: `for ((a=1; a <= LIMIT; a++))` Ответ: Синтаксис конструкции `for ((a=1; a <= LIMIT; a++))` верен.
7. Сравните язык `bash` с какими-либо языками программирования. Какие преимущества у `bash` по сравнению с ними? Какие недостатки? Ответ: 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 оптимизи-

рующим 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).