

Отчёт по лабораторной работе №12

Операционные системы

Сячинова Ксения Ивановна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Ответы на контрольные вопросы	13
4	Выводы	16

Список иллюстраций

2.1	Создание файла	6
2.2	Скрипт 1	7
2.3	Проверка скрипта 1	8
2.4	Изменённый скрипт	8
2.5	Проверка изменённого скрипта	9
2.6	Реализация команды map	9
2.7	Скрипт 2	10
2.8	Проверка скрипта 2	10
2.9	Скрипт 3	11
2.10	Проверка 3 скрипта	12

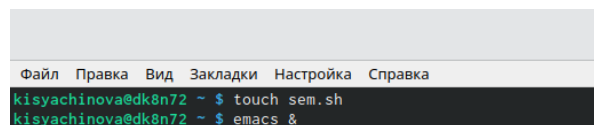
Список таблиц

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. Напишем командный файл, реализующий упрощённый механизм семафоров. Командный файл должен будет 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). Для этого создадим файл: `sem.sh` и напомним скрипт. (рис. 2.1), (рис. 2.2)



The image shows a terminal window with a menu bar at the top containing 'Файл', 'Правка', 'Вид', 'Закладки', 'Настройка', and 'Справка'. Below the menu bar, the terminal displays two lines of text: 'kisyachinova@dk8n72 ~ \$ touch sem.sh' and 'kisyachinova@dk8n72 ~ \$ emacs &'. The first line shows the command to create a file named 'sem.sh' using the 'touch' command. The second line shows the command to open an editor named 'emacs' in the background using the ampersand symbol '&'.

Рис. 2.1: Создание файла

```

1 #!/bin/bash
2 t1=$1
3 t2=$2
4 s1=$(date +%s)
5 s2=$(date +%s)
6 ((t=$s2-$s1))
7 while ((t < t1))
8 do
9     echo "Ожидайте"
10    sleep 1
11    s2=$(date +%s)
12    ((t=$s2-$s1))
13 done
14 s1=$(date +%s)
15 s2=$(date +%s)
16 ((t=$s2-$s1))
17 while ((t < t2))
18 do
19     echo "Выполнение"
20    sleep 1
21    s2=$(date +%s)
22    ((t=$s2-$s1))
23 done

```

Рис. 2.2: Скрипт 1

Далее проверим работу написанного скрипта с помощью команды “./sem.sh 4 7”, но перед этим добавим право на выполнение командой “chmod +x sem.sh”. Скрипт работает верно. (рис. 2.3)

```
kisyachinova@dk8n72 ~ $ chmod +x sem.sh
kisyachinova@dk8n72 ~ $ ./sem.sh 4 7
Ожидайте
Ожидайте
Ожидайте
Ожидайте
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
kisyachinova@dk8n72 ~ $
```

Рис. 2.3: Проверка скрипта 1

Далее изменим скрипт так, чтобы его можно было выполнять в нескольких терминалах.(рис. 2.4)

```
1#!/bin/bash
2function ogidanie
3{
4s1=$(date +%s")
5s2=$(date +%s")
6((t=$s2-$s1))
7while ((t < t1))
8do
9    echo "Ожидание"
10    sleep 1
11    s2=$(date +%s")
12    ((t=$s2-$s1))
13done
14}
15function vipolnenie
16{
17s1=$(date +%s")
18s2=$(date +%s")
19((t=$s2-$s1))
20while ((t < t2))
21do
22    echo "Выполнение"
23    sleep 1
24    s2=$(date +%s")
25    ((t=$s2-$s1))
26done
27}
28t1=$1
29t2=$2
30command=$3
31while true
32do
33    if [ "$command" == "Выход" ]
34    then
35        echo "Выход"
36        exit 0
37    fi
38    if [ "$command" == "Ожидание" ]
39    then
40        ogidanie
41    fi
42    if [ "$command" == "Выполнение" ]
43    then
44        vipolnenie
45    fi
46    echo "Следующее действие: "
47    read command
```

Рис. 2.4: Изменённый скрипт

Проверим работу с помощью команды “./sem.sh 2 3 Ожидание>/dev/pts/1 &”. Проверить работа данного скрипта не удалось, так как было отказано в доступе.(рис. 2.5)


```
[1]+ Выход 1      ./sem.sh 2 3 Ожидание > /dev/pts/1
kisyachinova@dk8n72 ~ $ ./sem.sh 2 3 Ожидание>/dev/pts/2 &
[1] 9930
kisyachinova@dk8n72 ~ $ bash: /dev/pts/2: Отказано в доступе

[1]+ Выход 1      ./sem.sh 2 3 Ожидание > /dev/pts/2
kisyachinova@dk8n72 ~ $ ./sem.sh 2 3 Выполнение > /dev/pts/1 &
[1] 9956
kisyachinova@dk8n72 ~ $ bash: /dev/pts/1: Отказано в доступе

[1]+ Выход 1      ./sem.sh 2 3 Выполнение > /dev/pts/1
kisyachinova@dk8n72 ~ $ ./sem.sh 2 3 Выполнение > /dev/pts/2 &
[1] 9959
kisyachinova@dk8n72 ~ $ bash: /dev/pts/2: Отказано в доступе

[1]+ Выход 1      ./sem.sh 2 3 Выполнение > /dev/pts/2
kisyachinova@dk8n72 ~ $
```

Рис. 2.5: Проверка изменённого скрипта

2. Реализуем команду (man с) помощью командного файла. Изучим 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. 2.6)

```
dropuser.1.bz2      mysql_find_rows.1.bz2
dt2dv.1.bz2         mysql_fix_extensions.1.bz2
dtd2xsd.1.bz2       mysqlhotcopy.1.bz2
dtrace.1.bz2        mysqlimport.1.bz2
dtsdec.1.bz2        mysql_install_db.1.bz2
du.1.bz2            mysql_ldb.1.bz2
dubdv.1.bz2         mysql_plugin.1.bz2
dumpeif.1.bz2       mysql_secure_installation.1.bz2
dumpiso.1.bz2       mysql.server.1.bz2
dumpkeys.1.bz2      mysql_setpermission.1.bz2
dump_xsettings.1.bz2 mysqlshow.1.bz2
dv2dt.1.bz2         mysqlslap.1.bz2
dvconnect.1.bz2     mysql-stress-test.pl.1.bz2
dvcont.1.bz2        mysqltest.1.bz2
dvi2fax.1.bz2       mysqltest_embedded.1
dvi2tty.1.bz2       mysql-test-run.pl.1.bz2
dvibook.1.bz2       mysql_tzinfo_to_sql.1.bz2
dviconcat.1.bz2     mysql_upgrade.1.bz2
dvicopy.1.bz2       mysql_waitpid.1.bz2
dvidvi.1.bz2        mzip.1.bz2
dvigif.1.bz2        namei.1.bz2
dvihp.1.bz2         nano.1.bz2
dvilj.1.bz2         nasm.1.bz2
dvilj2p.1           natspec.1.bz2
dvilj4.1            nautilus.1.bz2
dvilj4l.1           nautilus-autorun-software.1.bz2
dvilj6.1            nautilus-sendto.1.bz2
dvilualatex-dev.1  nc.1.bz2
dviluatex.1         ncat.1.bz2
dwindf.1.bz2        nccopy.1.bz2
```

Рис. 2.6: Реализация команды man

Далее я создала файл “man.sh”, в котором буду писать следйующий скрипт. (рис. 2.7)

```

report.md
1 #!/bin/bash
2 c=$1
3 if [ -f /usr/share/man/man1/$c.l.gz ]
4 then
5     gunzip -c /usr/share/man/man1/$1.l.gz | less
6 else
7     echo "Справки по данной команде нет"
8 fi

```

Рис. 2.7: Скрипт 2

Проверим работу скрипта, дадим ему право на выполнение “chmod +x man.sh” и проверим с помощью команды “./man.sh ls”, “./man.sh mkdir”(рис. 2.8)

```

kisyachinova@dk8n72 ~ $ chmod +x man.sh
kisyachinova@dk8n72 ~ $ ./man.sh ls
Справки по данной команде нет
kisyachinova@dk8n72 ~ $ ./man.sh mkdir
Справки по данной команде нет
kisyachinova@dk8n72 ~ $

```

Рис. 2.8: Проверка скрипта 2

- Используем встроенную переменную \$RANDOM, напишем командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтём, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767. Создадим файл для написания третьего скрипта (random.sh).(рис. 2.9)

```

1 #!/bin/bash
2 k=$1
3 for (( i=0; i<$k; i++ ))
4 do
5     (( char=$RANDOM%26+1 ))
6     case $char in
7         1) echo -n a;;
8         2) echo -n b;;
9         3) echo -n c;;
10        4) echo -n d;;
11        5) echo -n e;;
12        6) echo -n f;;
13        7) echo -n g;;
14        8) echo -n h;;
15        9) echo -n g;;
16        10) echo -n j;;
17        11) echo -n k;;
18        12) echo -n l;;
19        13) echo -n m;;
20        14) echo -n n;;
21        15) echo -n o;;
22        16) echo -n p;;
23        17) echo -n q;;
24        18) echo -n r;;
25        19) echo -n s;;
26        20) echo -n t;;
27        21) echo -n u;;
28        22) echo -n v;;
29        23) echo -n w;;
30        24) echo -n x;;
31        25) echo -n y;;
32        26) echo -n z;;
33    esac
34 done
35 echo
36
37

```

Рис. 2.9: Скрипт 3

Затем даём право на выполнение и с помощью команды “./random 5” и

“./random 12” проверяем выполнение работы.(рис. 2.10)

```
kisyachinova@dk8n72 ~ $ chmod +x random.sh
kisyachinova@dk8n72 ~ $ ./random.sh 5
qhvbj
kisyachinova@dk8n72 ~ $ ./random.sh 12
tzazpzfjwgtz
kisyachinova@dk8n72 ~ $
```

Рис. 2.10: Проверка 3 скрипта

3 Ответы на контрольные вопросы

1. `while [$1 != "exit"]` В данной строчке допущены следующие ошибки: • не хватает пробелов после первой скобки [и перед второй скобкой] • выражение `$1` необходимо взять в `"`, потому что эта переменная может содержать пробелы. Таким образом, правильный вариант должен выглядеть так: `while ["$1" != "exit"]`
2. Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами:
 - Первый: `VAR1="Hello, "VAR2=" World" VAR3="␣␣␣1VAR2" echo "$VAR3"`
Результат: Hello, World
 - Второй: `VAR1="Hello," VAR1+=" World" echo "$VAR1"` Результат: Hello, World
3. Команда `seq` в Linux используется для генерации чисел от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT. Параметры:
 - `seq LAST`: если задан только один аргумент, он создает числа от 1 до LAST с шагом шага, равным 1. Если LAST меньше 1, значение `is` не выдает.
 - `seq FIRST LAST`: когда заданы два аргумента, он генерирует числа от FIRST до LAST с шагом 1, равным 1. Если LAST меньше FIRST, он не выдает никаких выходных данных.
 - `seq FIRST INCREMENT LAST`: когда заданы три аргумента, он генерирует числа от FIRST до LAST на шаге INCREMENT. Если LAST меньше, чем FIRST, он не производит вывод.

- `seq -f «FORMAT» FIRST INCREMENT LAST`: эта команда используется для генерации последовательности в форматированном виде. `FIRST` и `INCREMENT` являются необязательными.
 - `seq -s «STRING» ПЕРВЫЙ ВКЛЮЧЕНО`: Эта команда используется для `STRING` для разделения чисел. По умолчанию это значение равно `/n`. `FIRST` и `INCREMENT` являются необязательными.
 - `seq -w FIRST INCREMENT LAST`: эта команда используется для выравнивания ширины путем заполнения начальными нулями. `FIRST` и `INCREMENT` являются необязательными.
4. Результатом данного выражения `$(10/3)` будет 3, потому что это целочисленное деление без остатка.
 5. Отличия командной оболочки `zsh` от `bash`:
 - В `zsh` более быстрое автодополнение для `cd` помощью `Tab`
 - В `zsh` существует калькулятор `zcalc`, способный выполнять вычисления внутри терминала
 - В `zsh` поддерживаются числа с плавающей запятой
 - В `zsh` поддерживаются структуры данных «хэш»
 - В `zsh` поддерживается раскрытие полного пути на основе неполных данных
 - В `zsh` поддерживается замена части пути
 - В `zsh` есть возможность отображать разделенный экран, такой же как разделенный экран `vim`
 6. `for((a=1; a<= LIMIT; a++))` синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать `$` перед переменными `()`.
 7. Преимущества скриптового языка `bash`:
 - Один из самых распространенных и ставится по умолчанию в большинстве дистрибутивах Linux, MacOS

- Удобное перенаправление ввода/вывода
 - Большое количество команд для работы с файловыми системами Linux
 - Можно писать собственные скрипты, упрощающие работу в Linux
- Недостатки скриптового языка bash:
- Дополнительные библиотеки других языков позволяют выполнить больше действий
 - Bash не является языком общего назначения
 - Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на скорости выполнения этого скрипта
 - Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий

4 Выводы

В ходе выполнения 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.