Отчёт по лабораторной работе №12

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Дарья Сергеевна Кочина

Содержание

Список иллюстраций

1 Цель работы

Целью данной лабораторной работы является изучение основ программирования в оболочке ОС UNIX. А также приобретение практических навыков написания более сложных командных файлов с использованием логических управляющих конструкций и циклов.

2 Задание

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

3 Теоретическое введение

Преимущества и недостатки Bash:

Многие языки программирования намного удобнее и понятнее для пользователя. Например, Python более быстр, так как компилируется байтами. Однако главное преимущество Bash – его повсеместное распространение. Более того, Bash позволяет очень легко работать с файловой системой без лишних конструкций (в отличие от других языков программирования). Но относительно таких bash очень сжат. То есть, например, C имеет гораздо более широкие возможности для разработчика.

4 Выполнение лабораторной работы

1. Написала 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). Для данной задачи я создала файл: sem.sh и написала соответствующий скрипт. (рис. [4.1], [4.2])

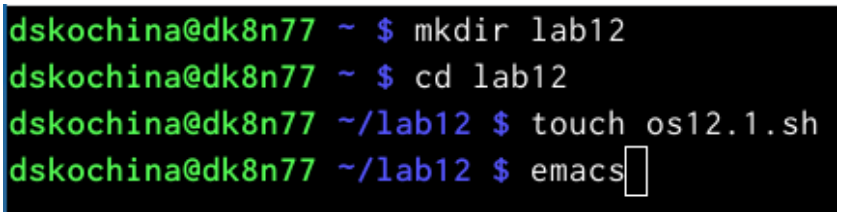

Рис. 4.1: Создание файла

```
#!/bin/bash
t1=$1
t2=$2
s1=$(date +"%s")
s2=$(date +"%s")
((t=$s2-$s1))
while ((t < t1))
do
    echo "Ожидайте"
    sleep 1
    s2=$(date +"%s")
    ((t=$s2-$s1))
done
s1=$(date +"%s")
s2=$(date +"%s")
((t=$s2-$s1))
while (( t < t2))
do
```

Рис. 4.2: Скрипт №1

```
    s2=$(date +"%s")
    ((t=$s2-$s1))
done
s1=$(date +"%s")
s2=$(date +"%s")
((t=$s2-$s1))
while (( t < t2))
do
    echo "Выполнение"
    sleep 1
    s2=$(date +"%s")
    ((t=$s2-$s1))
done
```

Рис. 4.3: Скрипт №1

2. Далее я проверила работу написанного скрипта, предварительно добавив право на исполнение файла. Скрипт работает корректно. (рис. [4.4])

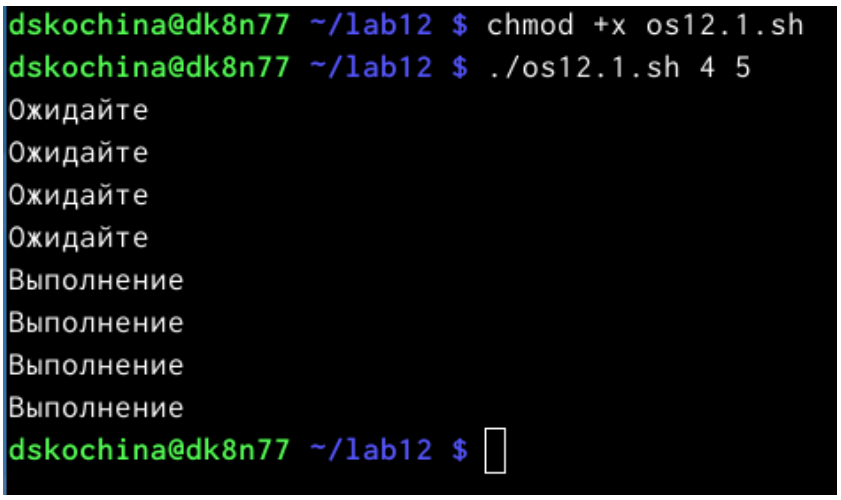

Рис. 4.4: Проверка работы скрипта

3. После этого я изменила скрипт так, чтобы его можно было выполнять в нескольких терминалах и проверила его работу. Однако у меня не получилось проверить работу скрипта, так как было отказно в доступе. (рис. [4.5], [4.6], [4.7], [4.8])

```
#!/bin/bash
function ogidania
{
s1=$(date +"%s")
s2=$(date +"%s")
((t=$s2-$s1))
while ((t < t1))
do
    echo "Ожидайте"
    sleep 1
    s2=$(date +"%s")
    ((t=$s2-$s1))
done
}
function vipolnenie
{
s1=$(date +"%s")
s2=$(date +"%s")
((t=$s2-$s1))
while (( t < t2))
```

Рис. 4.5: Изменённый скрипт №1

```
while (( t < t2))
do
    echo "Выполнение"
    sleep 1
    s2=$(date +"%s")
    ((t=$s2-$s1))
done
}
t1=$s1
t2=$s2
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then ogidanie
```

Рис. 4.6: Изменённый скрипт №1

```
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then ogidanie
    fi
    if [ "$command" == "Выполнение"
    then vipolnenie
    fi
    echo "Следующее действие: "
    read command
    done
```

Рис. 4.7: Изменённый скрипт №1

```
dskochina@dk8n77 ~/lab12 $ chmod +x os12.1.sh
dskochina@dk8n77 ~/lab12 $ ./os12.1.sh 2 3 Ожидание > /dev/pts/1 &
[1] 6526
dskochina@dk8n77 ~/lab12 $ bash: /dev/pts/1: Отказано в доступе
dskochina@dk8n77 ~/lab12 $ ./os12.1.sh 2 3 Ожидание > /dev/pts/2
bash: /dev/pts/2: Отказано в доступе
[1]+  Выход 1             ./os12.1.sh 2 3 Ожидание > /dev/pts/1
dskochina@dk8n77 ~/lab12 $ ./os12.1.sh 2 5 Выполнение > /dev/pts/2 &
[1] 6787
dskochina@dk8n77 ~/lab12 $ bash: /dev/pts/2: Отказано в доступе
```

Рис. 4.8: Проверка работы скрипта

4. 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 ls сразу же просмотрев содержимое справки. Командный файл должен получать в виде аргумента командной строки название команды и в виде результата выда-

вать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1. (рис. [4.9], [4.10], [4.11])

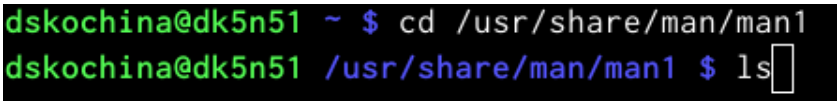

Рис. 4.9: Реализация команды man

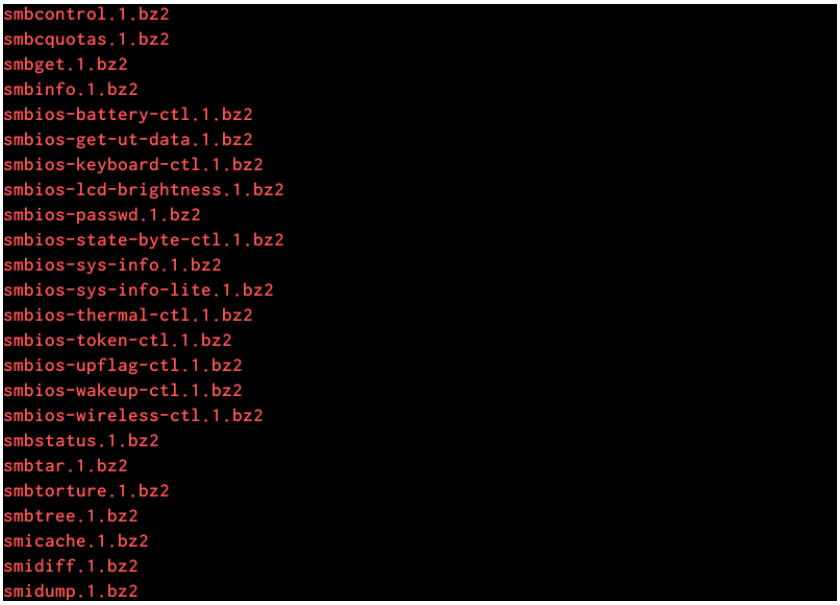

Рис. 4.10: Реализация команды man

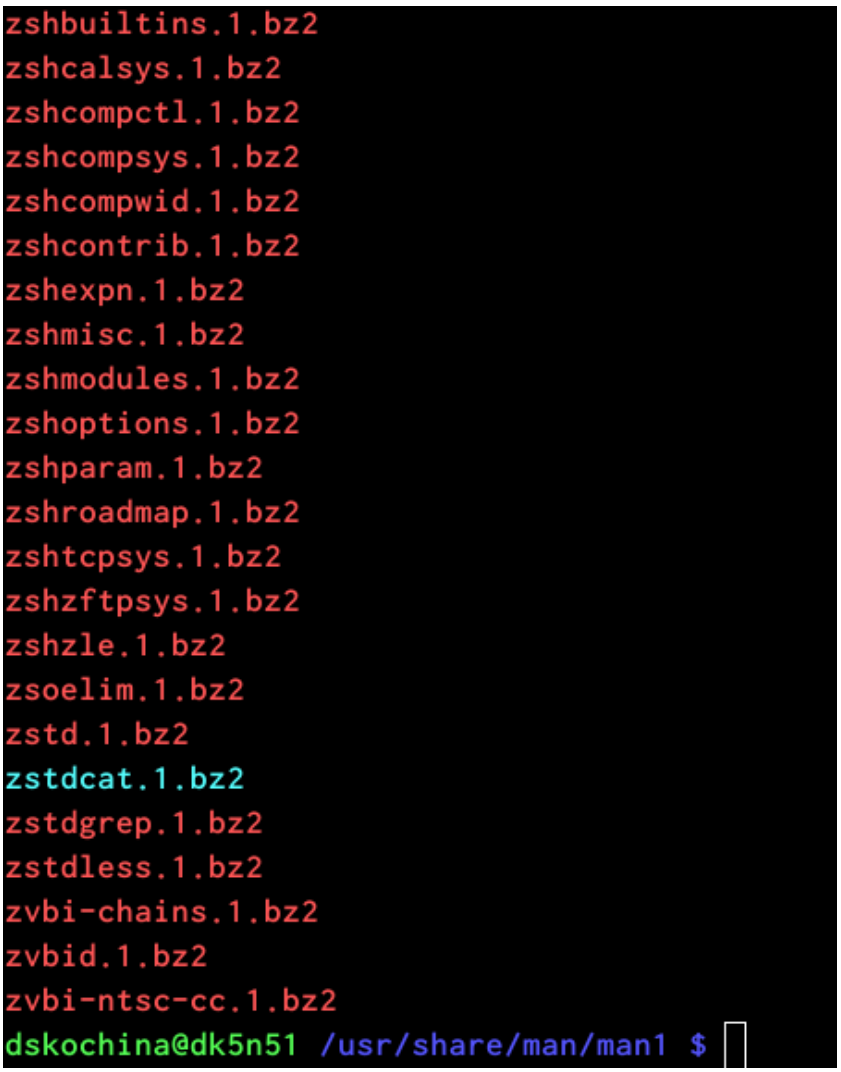

Рис. 4.11: Реализация команды man

5. Для данной задачи я создала файл и написала соответствующий скрипт. (рис. [4.12])

```
#!/bin/bash
c=$1
if [ -f /usr/share/man/man1/$c.1.gz ]
then
    gunzip -c /usr/share/man/man1/$1.1.gz | less
else
    echo "Справка по данной команде нет"
fi
```

Рис. 4.12: Скрипт №2

6. Далее я проверила работу написанного скрипта, предварительно добавив право на исполнение файла. Скрипт работает корректно. (рис. [4.13])

```
dskochina@dk5n51 ~ $ cd lab12
dskochina@dk5n51 ~/lab12 $ emacs
dskochina@dk5n51 ~/lab12 $ touch os12.2.sh
dskochina@dk5n51 ~/lab12 $ emacs
dskochina@dk5n51 ~/lab12 $ chmod +x os12.2.sh
dskochina@dk5n51 ~/lab12 $ ./os12.2.sh ls
Справка по данной команде нет
dskochina@dk5n51 ~/lab12 $ ./os12.2.sh mkdir
Справка по данной команде нет
dskochina@dk5n51 ~/lab12 $
```

Рис. 4.13: Проверка работы скрипта

7. Используя встроенную переменную $RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. Для данной задачи я создала файл и написала соответствующий скрипт. (рис. [4.14], [4.15])

```
#!/bin/bash
k=$1
for (( i=0; i<$k; i++ ))
do
    (( char=$RANDOM%26+1 ))
    case $char in
        1) echo -n a;;
        2) echo -n b;;
        3) echo -n c;;
        4) echo -n d;;
        5) echo -n e;;
        6) echo -n f;;
        7) echo -n g;;
        8) echo -n h;;
        9) echo -n i;;
        10) echo -n j;;
        11) echo -n k;;
        12) echo -n l;;
        13) echo -n m;;
        14) echo -n n;;
```

Рис. 4.14: Скрипт №3

```
        14) echo -n n;;
        15) echo -n o;;
        16) echo -n p;;
        17) echo -n q;;
        18) echo -n r;;
        19) echo -n s;;
        20) echo -n t;;
        21) echo -n u;;
        22) echo -n v;;
        23) echo -n w;;
        24) echo -n x;;
        25) echo -n y;;
        26) echo -n z;;
    esac
done
echo
```

Рис. 4.15: Скрипт №3

8. Далее я проверила работу написанного скрипта, предварительно добавив право на исполнение файла. Скрипт работает корректно. (рис. [4.16])

```
dskochina@dk5n51 ~ $ chmod +x os12.3.sh
dskochina@dk5n51 ~ $ ./os12.3.sh 5
yxici
dskochina@dk5n51 ~ $ ./os12.3.sh 10
llwuqwcirv
dskochina@dk5n51 ~ $
```

Рис. 4.16: Проверка работы скрипта

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=“$VAR1VAR2”

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 zshot bash:

В zsh более быстрое автодополнение для cdc 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 поддерживаетсязаменачастипути

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 умолчаниюв 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 языков общего назначения

Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на быстроте выполнения этого скрипта

Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий.

5 Выводы

В ходе выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX. А также приобрела практические навыки написания более сложные командных файлов с использованием логических управляющих конструкций и циклов.