

# **Отчёт по лабораторной работе №14**

**Операционные системы**

Пономарева Татьяна Александровна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>4</b>	<b>Ответы на контрольные вопросы</b>	<b>12</b>
<b>5</b>	<b>Выводы</b>	<b>15</b>
	<b>Список литературы</b>	<b>16</b>

# Список иллюстраций

3.1	Исполнение 1	8
3.2	Исполнение 2	10
3.3	Исполнение 3	11

## **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 Теоретическое введение

Командный процессор, или оболочка (shell), является важной частью операционных систем UNIX и Linux, позволяя пользователю взаимодействовать с системой через командную строку. Оболочка позволяет не только выполнять простые команды, но и писать сложные сценарии (скрипты), которые могут включать в себя различные управляющие конструкции и функциональные элементы, такие как условия, циклы и функции.

### 3 Выполнение лабораторной работы

1. Написать командный файл, реализующий упрощённый механизм семафоров. Для реализации упрощённого механизма семафоров можно использовать файловые блокировки. Мы будем использовать файл в качестве “семафора”, чтобы сигнализировать, что ресурс занят или свободен.

```
#!/bin/bash
```

```
# Устанавливаем имя файла для семафора
```

```
SEMAPHORE="/tmp/semaphore.lock"
```

```
# Время ожидания
```

```
t1=5
```

```
t2=3
```

```
# Функция для ожидания освобождения ресурса
```

```
wait_for_resource() {
```

```
    while [ -e "$SEMAPHORE" ]; do
```

```
        echo "Ресурс занят, ожидаем..."
```

```
        sleep $t1
```

```
    done
```

```
}
```

```
# Функция для использования ресурса
```

```

use_resource() {
    touch "$SEMAPHORE"
    echo "Ресурс используется..."
    sleep $t2
    rm -f "$SEMAPHORE"
    echo "Ресурс освобожден."
}

```

```

# Основная логика
wait__resource
use_resource

```

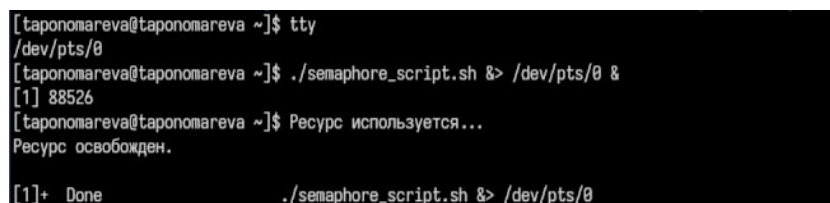
Делаю файл исполняемым при помощи команды: `chmod +x ~/semaphore_script.sh`

Запуск скрипта в фоновом режиме: `./semaphore_script.sh &> /dev/tty# &` (где # — это номер терминала)

Этот скрипт проверяет, занят ли ресурс, и если да, то ждёт, пока он не будет освобожден. После этого скрипт использует ресурс, а затем освобождает его.

Для взаимодействия трёх и более процессов можно создать дополнительные экземпляры этого скрипта и использовать различные файлы для каждого процесса.

Исполнение (рис. 3.1).



```

[taonomareva@taonomareva ~]$ tty
/dev/pts/8
[taonomareva@taonomareva ~]$ ./semaphore_script.sh &> /dev/pts/8 &
[1] 88526
[taonomareva@taonomareva ~]$ Ресурс используется...
Ресурс освобожден.
[1]+ Done ./semaphore_script.sh &> /dev/pts/8

```

Рис. 3.1: Исполнение 1

2. Реализовать команду `man` с помощью командного файла. Этот скрипт будет принимать команду, искать справку в каталоге `/usr/share/man/man1` и показывать содержимое с помощью `less` (если справка найдена).



```
#!/bin/bash
```

```
# Проверка аргументов
```

```
if [ $# -ne 1 ]; then
```

```
    echo "Использование: $0 <команда>"
```

```
    exit 1
```

```
fi
```

```
COMMAND=$1
```

```
MAN_PAGE="/usr/share/man/man1/$COMMAND.1.gz"
```

```
# Проверка существования файла справки
```

```
if [ -f "$MAN_PAGE" ]; then
```

```
    echo "Открытие справки по команде '$COMMAND'..."
```

```
    zless "$MAN_PAGE"
```

```
else
```

```
    echo "Справка для команды '$COMMAND' не найдена."
```

```
fi
```

Делаю файл исполняемым при помощи команды: `chmod +x ~/lab14_2.sh`

Пример использования: `./lab14_2.sh ls`

Скрипт проверяет наличие справочного файла для указанной команды в каталоге `/usr/share/man/man1` и, если файл существует, открывает его с помощью команды `zless`.

Исполнение (рис. 3.2).

```

[taonomareva@taonomareva ~]$ ~/lab14_2.sh ls
Открытие справки по команде 'ls'...
.\\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.48.5.
.TH LS "1" "November 2024" "GNU coreutils 9.5" "User Commands"
.SH NAME
ls \- list directory contents
.SH SYNOPSIS
.B ls
[\fI\,OPTION\ /\fR]... [\fI\,FILE\ /\fR]...

```

Рис. 3.2: Исполнение 2

3. Генерация случайной последовательности букв латинского алфавита с использованием \$RANDOM. Для этого можно воспользоваться переменной \$RANDOM для генерации случайных чисел и преобразовывать их в буквы.

```
#!/bin/bash
```

```
# Длина последовательности
```

```
LENGTH=10
```

```
# Генерация случайной последовательности букв
```

```
generate_random_string() {
```

```
    local result=""
```

```
    for ((i=0; i<$LENGTH; i++)); do
```

```
        # Генерация случайной буквы
```

```
        letter=$((RANDOM % 26 + 65)) # Число от 65 до 90 (A-Z)
```

```
        result+=$(printf "\\$(printf '%03o' $letter)")
```

```
    done
```

```
    echo "$result"
```

```
}
```

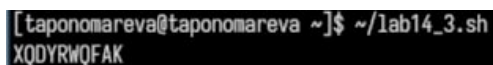
```
# Вывод случайной строки
```

```
generate_random_string
```

Делаю файл исполняемым при помощи команды: `chmod +x ~/lab14_3.sh`

Пример использования: ./lab14\_3.sh Этот скрипт генерирует случайную строку длиной 10 символов, используя \$RANDOM для генерации случайных чисел в диапазоне от 0 до 25 (для латинских букв A-Z).

Исполнение (рис. 3.3).



```
[taonomareva@taonomareva ~]$ ./lab14_3.sh  
XQDYRWQFAK
```

Рис. 3.3: Исполнение 3

## 4 Ответы на контрольные вопросы

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

Ошибка: Пропущены пробелы вокруг оператора сравнения. Правильный синтаксис:

```
while [ "$1" != "exit" ]
```

2. Как объединить (конкатенация) несколько строк в одну? В Bash можно объединить строки несколькими способами:

Использовать символ + (внутри кавычек):

```
str1="Hello"
str2="world"
result="$str1 $str2"
echo "$result" # Вывод: Hello world
```

Использовать команду `echo` с конкатенацией:

```
echo "Hello" "world" # Вывод: Hello world
```

3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать её функционал при программировании на `bash`? Утилита `seq` используется для генерации последовательностей чисел, например:

`seq 1 5` # Выводит: 1 2 3 4 5 Иными способами можно сгенерировать такую последовательность с помощью цикла `for`:

```
for i in {1..5}; do echo $i; done # Выводит: 1 2 3 4 5
```

4. Какой результат даст вычисление выражения  $\$(10/3)$ ? Результат будет равен 3, так как это целочисленное деление в Bash. При выполнении деления целых чисел в Bash результат округляется в меньшую сторону.

5. Укажите кратко основные отличия командной оболочки zsh от bash.

zsh обладает более мощными возможностями автодополнения команд и путей, чем bash.

В zsh есть поддержка тем и плагинов, что делает её более настраиваемой.

zsh предлагает расширенные возможности работы с массивами и строками.

В zsh присутствует более гибкое управление историей команд.

В отличие от bash, zsh поддерживает “glob” с возможностью использования регулярных выражений для сопоставления файлов.

6. Проверьте, верен ли синтаксис данной конструкции:

for ((a=1; a <= LIMIT; a++)) Ответ: Синтаксис верен при условии, что LIMIT является заранее определённой переменной с числовым значением. Важно, чтобы переменная LIMIT была инициализирована до начала цикла.

7. Сравните язык bash с какими-либо языками программирования. Какие преимущества у bash по сравнению с ними? Какие недостатки?

Преимущества bash:

Простота в написании сценариев для автоматизации задач в ОС UNIX/Linux.

Поддержка выполнения команд непосредственно в оболочке и взаимодействие с операционной системой.

Лёгкость в использовании для написания одноразовых скриптов и утилит для системного администрирования.

Недостатки bash:

Отсутствие строгой типизации данных.

Ограниченные возможности для сложных вычислений и работы с большими объёмами данных по сравнению с полноценными языками программирования.

Сложности при отладке, так как ошибки часто трудно выявить в больших скриптах.

## **5 Выводы**

Были изучены основы программирования в оболочке ОС UNIX. Были получены знания о написании более сложных командных файлов с использованием логических управляющих конструкций и циклов.

# **Список литературы**

1. Курс на ТУИС