

Лабораторная работа №14

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

Полякова Юлия Александровна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Контрольные вопросы	12
5	Вывод	16

Список иллюстраций

3.1	Листинг semaphore.sh	7
3.2	Листинг launcher.sh	8
3.3	Запуск launcher.sh	8
3.4	Листинг my_man.sh	9
3.5	Запуск my_man.sh	9
3.6	Справка после запуска	10
3.7	Листинг random_letters.sh	11
3.8	Запуск random_letters.sh	11

Список таблиц

1 Цель работы

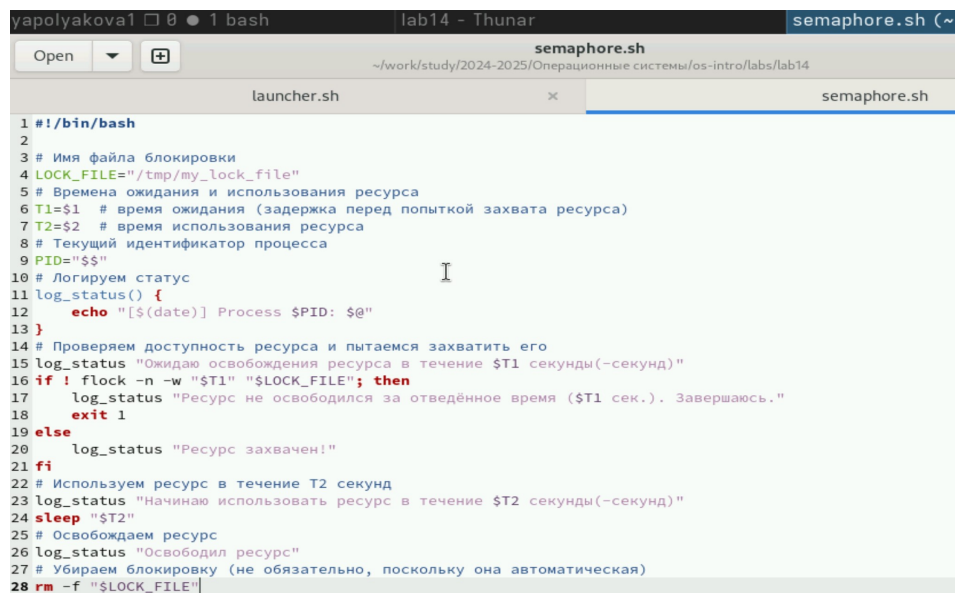
Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

1. Ознакомиться с теоретическим материалом.
2. Написать программы.
3. Ответить на контрольные вопросы.

3 Выполнение лабораторной работы

1. Рассмотрим первый скрипт по заданию: Написать командный файл, реализующий упрощённый механизм семафоров. Доработать программу так, чтобы имела возможность взаимодействия трёх и более процессов.(рис. 3.1).



```
#!/bin/bash
# Имя файла блокировки
LOCK_FILE="/tmp/my_lock_file"
# Времена ожидания и использования ресурса
T1=$1 # время ожидания (задержка перед попыткой захвата ресурса)
T2=$2 # время использования ресурса
# Текущий идентификатор процесса
PID="$@"
# Логирование статуса
log_status() {
    echo "[$(date)] Process $PID: $@"
}
# Проверяем доступность ресурса и пытаемся захватить его
log_status "Ожидаю освобождения ресурса в течение $T1 секунды(-секунд)"
if ! flock -n -w "$T1" "$LOCK_FILE"; then
    log_status "Ресурс не освободился за отведённое время ($T1 сек.). Завершаюсь."
    exit 1
else
    log_status "Ресурс захвачен!"
fi
# Используем ресурс в течение T2 секунд
log_status "Начинаю использовать ресурс в течение $T2 секунды(-секунд)"
sleep "$T2"
# Освобождаем ресурс
log_status "Освободил ресурс"
# Убираем блокировку (не обязательно, поскольку она автоматическая)
rm -f "$LOCK_FILE"
```

Рис. 3.1: Листинг semaphore.sh

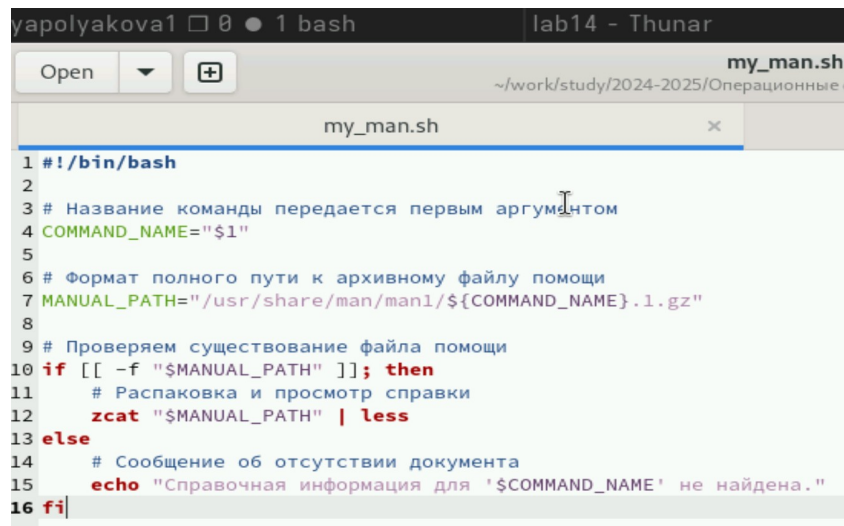
2. Для запуска пишем отдельную программу (рис. 3.2)

Рис. 3.2: Листинг launcher.sh

3. Запускаем первый скрипт так (рис. 3.3)

Рис. 3.3: Запуск launcher.sh

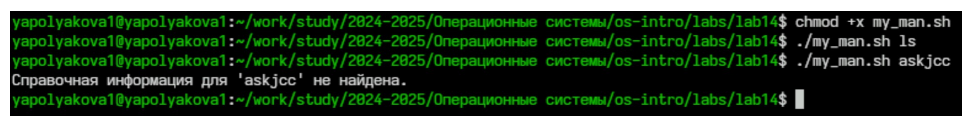
4. Рассмотрим второе задание: Реализовать команду `map` с помощью командного файла. (рис. 3.4)



```
1 #!/bin/bash
2
3 # Название команды передается первым аргументом
4 COMMAND_NAME="$1"
5
6 # Формат полного пути к архивному файлу помощи
7 MANUAL_PATH="/usr/share/man/man1/${COMMAND_NAME}.1.gz"
8
9 # Проверяем существование файла помощи
10 if [[ -f "$MANUAL_PATH" ]]; then
11     # Распаковка и просмотр справки
12     zcat "$MANUAL_PATH" | less
13 else
14     # Сообщение об отсутствии документа
15     echo "Справочная информация для '$COMMAND_NAME' не найдена."
16 fi
```

Рис. 3.4: Листинг my_man.sh

5. Запуск второго задания с существующей и несуществующей командой (рис. 3.5)



```
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab14$ chmod +x my_man.sh
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab14$ ./my_man.sh ls
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab14$ ./my_man.sh askjcc
Справочная информация для 'askjcc' не найдена.
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab14$
```

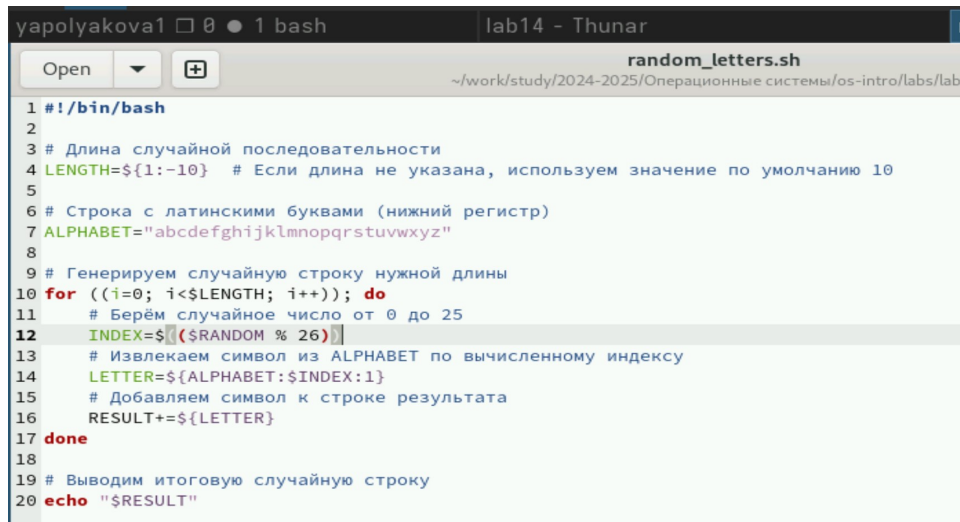
Рис. 3.5: Запуск my_man.sh

6. Открывается справка, если команда существует. Здесь ls (рис. 3.6)

```
yapolyakova1 0 1 bash | lab14 - Thunar
.\" 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]...
.SH DESCRIPTION
.\" Add any additional description here
.PP
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of \fB\--cftuvSUX\fR nor \fB\--sort\fR is specified.
.PP
Mandatory arguments to long options are mandatory for short options too.
.TP
\fB\--a\fR, \fB\--all\fR
do not ignore entries starting with .
.TP
\fB\--A\fR, \fB\--almost-all\fR
do not list implied . and ..
.TP
\fB\--author\fR
with \fB\--l\fR, print the author of each file
.TP
\fB\--b\fR, \fB\--escape\fR
print C\-\style escapes for nongraphic characters
.TP
\fB\--block-size=SIZE\fR=\fI\,SIZE\...\fR
with \fB\--l\fR, scale sizes by SIZE when printing them;
e.g., '\fB\--block-size=M\fR'; see SIZE format below
.TP
\fB\--B\fR, \fB\--ignore-backups\fR
do not list implied entries ending with ~
.TP
\fB\--c\fR
```

Рис. 3.6: Справка после запуска

7. Третье задание: Используя встроенную переменную \$RANDOM, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. (рис. 3.7)



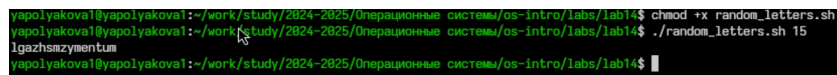
```

1 #!/bin/bash
2
3 # Длина случайной последовательности
4 LENGTH=${1:-10} # Если длина не указана, используем значение по умолчанию 10
5
6 # Строка с латинскими буквами (нижний регистр)
7 ALPHABET="abcdefghijklmnopqrstuvwxyz"
8
9 # Генерируем случайную строку нужной длины
10 for ((i=0; i<$LENGTH; i++)); do
11     # Берём случайное число от 0 до 25
12     INDEX=$((RANDOM % 26))
13     # Извлекаем символ из ALPHABET по вычисленному индексу
14     LETTER=${ALPHABET:$INDEX:1}
15     # Добавляем символ к строке результата
16     RESULT+=$LETTER
17 done
18
19 # Выводим итоговую случайную строку
20 echo "$RESULT"

```

Рис. 3.7: Листинг random_letters.sh

8. Как и со всеми предыдущими файлами, открываем доступ и запускаем (рис. 3.8)



```

yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab14$ chmod +x random_letters.sh
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab14$ ./random_letters.sh 15
lgazhsmzymentum
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab14$

```

Рис. 3.8: Запуск random_letters.sh

4 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке:

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

Ошибка: Пробелы и кавычки отсутствуют. Правильная форма должна выглядеть следующим образом:

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

2. Как объединить (конкатенация) несколько строк в одну?

Конкатенация строк осуществляется простым соединением переменных или строковых значений с использованием оператора + или же путем простого следования друг за другом.

Пример:

```
string="Hello "  
string+="World!"  
echo "$string"    # Output: Hello World!
```

Альтернативный способ — использование команды printf:

```
var1="Hello"  
var2="World"  
result=$(printf "%s%s\n" "$var1" "$var2")  
echo "$result"    # Output: HelloWorld
```

3. Утилита seq

Утилита **seq** используется для генерации последовательностей чисел. Она принимает аргументы, определяющие начальное значение, конечное значение и шаг последовательности.

Примеры использования:

- Последовательность от 1 до 10:

```
seq 1 10
```

- Последовательность от 1 до 10 с шагом 2:

```
seq 1 2 10
```

Альтернативные способы реализации функционала seq в Bash:

- Использование цикла for:

```
for i in {1..10}; do echo "$i"; done
```

- Используя цикл while:

```
i=1
while [ $i -le 10 ]; do
    echo "$i"
    let i++
done
```

4. Какой результат даст выражение '((10/3))'?

Результатом целочисленного деления является округленное вниз число. То есть:

```
$((10 / 3)) = 3
```

5. Основные отличия оболочек ZSH и BASH

- **ZSH:** Более расширенная функциональность по умолчанию, поддержка улучшенных автодополнений, псевдонимов, регулярных выражений прямо в шаблонах имен файлов, автоматическое исправление ошибок в командах (correc), встроенный механизм расширения путей (^, %) и т.п.
- **BASH:** Менее гибкая по функциональности, стандартная и широко используемая оболочка Unix-подобных операционных систем, доступная почти повсеместно.

Основные различия:

- **Автодополнение:** ZSH предлагает гораздо больше возможностей автоматического дополнения.
- **Коррекция ввода:** ZSH автоматически пытается исправить неправильно введенные команды.
- **Расширение путей:** ZSH поддерживает более удобные механизмы работы с путями.
- **Интерактивность:** ZSH имеет более развитые возможности интерактивного взаимодействия.

6. Проверка синтаксиса конструкции

```
for ((a=1; a <= LIMIT; a++))
```

Конструкция написана верно. Это правильный синтаксис циклов в стиле C в shell.

7. Сравнение Bash с другими языками программирования

Преимущества Bash:

- **Простота:** Bash удобен для написания простых скриптов автоматизации и управления системой.

- **Интеграция с Unix:** Легкость интеграции с файловыми системами, процессами и инструментами операционной системы.
- **Широкая доступность:** Доступен практически на всех системах Linux и Unix.
- **Скриптовая природа:** Позволяет быстро решать повседневные задачи администрирования и разработки.

Недостатки Bash:

- **Производительность:** Медленнее других языков программирования при сложных операциях и больших объемах данных.
- **Ограниченность типов данных:** Отсутствие поддержки классов, объектов и полноценных структур данных.
- **Отсутствие строгого контроля типов:** Ошибки легко пропустить из-за отсутствия проверки типов на этапе компиляции.
- **Недостаточная масштабируемость:** Сложно поддерживать большие проекты на Bash из-за ограничений среды исполнения.

5 Вывод

Были изучены основы программирования в оболочке ОС UNIX. Мы научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.