

Лабораторная работа №14

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Полякова Ю.А.

28 февраля 2007

Российский университет дружбы народов, Москва, Россия

Преподаватель Кулябов Д. С., д.ф.-м.н., профессор

Информация

- Полякова Юлия Александровна
- Студент
- Российский университет дружбы народов
- yulya.polyakova.07@mail.ru
- <https://github.com/JuliaMaffin123>



Вводная часть

- Умение работать с командными файлами важно для понимания системы и будет полезно в будущем

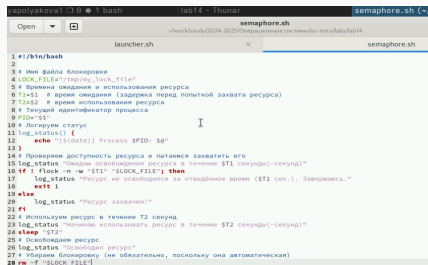
- командный процессор ОС UNIX. Командные файлы

- Цель: Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.
- Задачи:
 - Ознакомиться с теоретическим материалом.
 - Написать 3 командных файла по заданию.
 - Ответить на контрольные вопросы.

- редактор gedit
- терминал
- командные файлы

Выполнение лабораторной работы

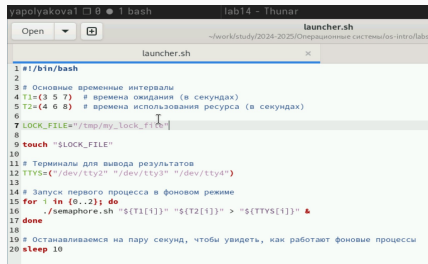
Рассмотрим первый скрипт по заданию: Написать командный файл, реализующий упрощённый механизм семафоров. Доработать программу так, чтобы имела возможность взаимодействия трёх и более процессов.



```
1 #!/bin/bash
2
3 # имя файла блокировки
4 LOCK_FILE="/tmp/my_lock_file"
5 # время ожидания и использования ресурса
6 T1=51 # время ожидания (задержка перед попыткой захвата ресурса)
7 T2=52 # время использования ресурса
8 # текущий идентификатор процесса
9 PID=$$
10 # Логируем статус
11 log_status() {
12     echo "[$(date)] Process $PID: $@"
13 }
14 # Проверим доступность ресурса и попытаемся захватить его
15 log_status "Ожидаем освобождения ресурса в течение $T1 секунды(-секунд)"
16 if ! flock -n "$T1" "$LOCK_FILE"; then
17     log_status "Ресурс не освободился за отведённое время ($T1 сек.). Завершаем."
18     exit 1
19 else
20     log_status "Ресурс захвачен!"
21 fi
22 # Используем ресурс в течение T2 секунд
23 log_status "Начинаю использовать ресурс в течение $T2 секунды(-секунд)"
24 sleep "$T2"
25 # Освобождаем ресурс
26 log_status "Освободил ресурс"
27 # Убираем блокировку (не обязательно, поскольку она автоматическая)
28 rm -f "$LOCK_FILE"
```

Рис. 1: Листинг semaphore.sh

Для запуска пишем отдельную программу



The screenshot shows a terminal window titled 'lab14 - Thunar' with a file manager interface. The file 'launcher.sh' is open, showing a shell script. The script includes comments in Russian and code for setting intervals, locking a file, and running a semaphore process in the background.

```
1 #!/bin/bash
2
3 # Основные временные интервалы
4 T1=(3 5 7) # времена ожидания (в секундах)
5 T2=(4 6 8) # времена использования ресурса (в секундах)
6
7 LOCK_FILE="/tmp/my_lock_file"
8
9 touch "$LOCK_FILE"
10
11 # Терминалы для вывода результатов
12 TTYS=("/dev/tty2" "/dev/tty3" "/dev/tty4")
13
14 # Запуск первого процесса в фоновом режиме
15 for i in {0..2}; do
16   ./semaphore.sh "${T1[i]}" "${T2[i]}" > "${TTYS[i]}" &
17 done
18
19 # Останавливаемся на пару секунд, чтобы увидеть, как работают фоновые процессы
20 sleep 10
```

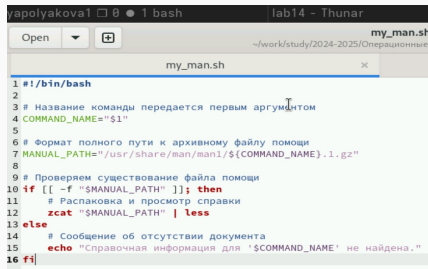
Рис. 2: Листинг launcher.sh

Запускаем первый скрипт так

```
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab14$ chmod +x semaphore.sh  
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab14$ chmod +x launcher.sh  
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab14$ sudo bash launcher.sh  
[sudo] password for yapolyakova1:
```

Рис. 3: Запуск launcher.sh

Рассмотрим второе задание: Реализовать команду man с помощью командного файла.



```
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. 4: Листинг my_man.sh

Запуск второго задания с существующей и несуществующей командой

```
ya@ya:~/work/study/2824-2825/Операционные системы/os-intro/labs/lab14$ chmod +x my_man.sh
ya@ya:~/work/study/2824-2825/Операционные системы/os-intro/labs/lab14$ ./my_man.sh ls
ya@ya:~/work/study/2824-2825/Операционные системы/os-intro/labs/lab14$ ./my_man.sh askjcc
Справочная информация для 'askjcc' не найдена.
ya@ya:~/work/study/2824-2825/Операционные системы/os-intro/labs/lab14$
```

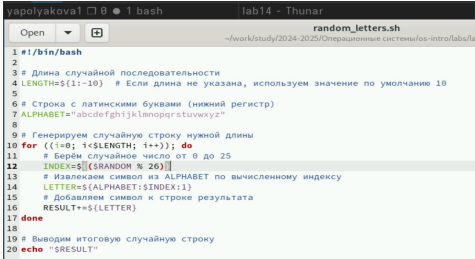
Рис. 5: Запуск my_man.sh

Открывается справка, если команда существует. Здесь ls

[illegible]

Рис. 6: Справка после запуска

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



```
#!/bin/bash
#
# Длина случайной последовательности
LENGTH=${1:-10} # Если длина не указана, используем значение по умолчанию 10
#
# Строка с латинскими буквами (нижний регистр)
ALPHABET="abcdefghijklmnopqrstuvwxyz"
#
# Генерируем случайную строку нужной длины
for ((i=0; i<$LENGTH; i++)); do
# Берём случайное число от 0 до 25
INDEX=$((RANDOM % 26))
# Извлекаем символ из ALPHABET по вычисленному индексу
LETTER=${ALPHABET:$INDEX:1}
# Добавляем символ к строке результата
RESULT+=$LETTER
done
# Выводим итоговую случайную строку
echo "$RESULT"
```

Рис. 7: Листинг random_letters.sh

Как и со всеми предыдущими файлами, открываем доступ и запускаем

```
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. 8: Запуск random_letters.sh

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 введённые 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.

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