

# 14

Рафи Кази ар  
НКАбд-03-24  
1032238132@pfur.ru

## **Задание 1: Упрощенный механизм семафоров**

Это задание реализует концепцию **мьютекса** (взаимоисключающей блокировки) с использованием файла в качестве семафора.

### **Листинг скрипта semaphore.sh:**

```
#!/bin/bash

# Аргументы: $1 - время ожидания (t1), $2 - время
#             использования (t2)
t1=$1
t2=$2
semaphore_file="/tmp/lab12_semaphore.lock"

# Функция для ожидания освобождения ресурса
wait_for_resource() {
    local wait_time=0
    # Проверяем существование файла-семафора каждую
    # секунду
    while [ -f "$semaphore_file" ] && [ $wait_time -lt $t1 ]; do
        echo "Процесс $$: Ресурс занят. Ожидание... ($wait_time
        сек.)"
        sleep 1
        ((wait_time++))
    done
```

```

# Если время ожидания вышло, а ресурс не освободился
if [ -f "$semaphore_file" ]; then
echo "Процесс $$: Время ожидания истекло! Ресурс не
получен."
exit 1
else
# Если ресурс свободен, занимаем его (создаем файл)
touch "$semaphore_file"
echo "Процесс $$: Ресурс получен! Использую в течение $t2
сек."
return 0
fi
}

# Функция для освобождения ресурса
release_resource() {
if [ -f "$semaphore_file" ]; then
rm -f "$semaphore_file"
echo "Процесс $$: Ресурс освобожден."
fi
}

# Главная логика скрипта

# Пытаемся занять ресурс
wait_for_resource

# Если заняли успешно (функция вернула код 0), то
используем его
if [ $? -eq 0 ]; then
# Используем ресурс заданное время
sleep $t2
# После использования освобождаем
release_resource
fi

```

### **Инструкция по запуску и скриншот:**

1. **Откройте два терминала** (например, tty2 и tty3). Узнать их имена можно командой tty.
2. **В первом терминале** запустите скрипт в

фоновом режиме, перенаправив вывод во второй терминал. Предположим, что первый терминал — /dev/pts/1, а второй — /dev/pts/2.

- # В терминале 1 (/dev/pts/1):  
chmod +x semaphore.sh  
./semaphore.sh 10 5 > /dev/pts/2 &

*Этот процесс начнет ждать освобождения ресурса 10 секунд и, получив его, будет использовать 5 секунд. Его вывод мы будем видеть во втором терминале.*

3. **Во втором терминале** сразу запустите скрипт в привилегированном (обычном) режиме.

- # В терминале 2 (/dev/pts/2):  
./semaphore.sh 10 5

*Этот процесс сразу создаст файл-блокировку и займет ресурс. Вывод обоих процессов будет виден в этом окне.*



krafi@pop-os:~/Documents/arciture rudn/14

```
[& krafi@pop-os ]--[⌚ 04:11 PM]--> www.krafi.info
└─ Documents/arciture rudn/14
-> nano semaphore.sh
```

```
[& krafi@pop-os ]--[⌚ 04:13 PM]--> www.krafi.info
└─ Documents/arciture rudn/14 took 17s
-> sudo chmod +x ./semaphore.sh
```

```
[& krafi@pop-os ]--[⌚ 04:13 PM]--> www.krafi.info
└─ Documents/arciture rudn/14
-> ./semaphore.sh 10 5 > /dev/pts/2 &
[1] 176469
zsh: permission denied: /dev/pts/2
[1] + 176469 exit 1 ./semaphore.sh 10 5 > /dev/pts/2
```

```
[& krafi@pop-os ]--[⌚ 04:14 PM]--> www.krafi.info
└─ Documents/arciture rudn/14
-> sudo !!
```

```
[& krafi@pop-os ]--[⌚ 04:14 PM]--> www.krafi.info
└─ Documents/arciture rudn/14
-> sudo ./semaphore.sh 10 5 > /dev/pts/2 &
[1] 176519
zsh: permission denied: /dev/pts/2
[1] + 176519 exit 1 sudo ./semaphore.sh 10 5 > /dev/pts/2
```

```
[& krafi@pop-os ]--[⌚ 04:14 PM]--> www.krafi.info
└─ Documents/arciture rudn/14
-> ./semaphore.sh 10 5 | sudo tee /dev/pts/2 > /dev/null &
[1] 176590 176591
tee: /dev/pts/2: Permission denied
```

```
[& krafi@pop-os ]--[⌚ 04:15 PM]--> www.krafi.info
└─ Documents/arciture rudn/14
->
[1] + 176590 done ./semaphore.sh 10 5 |
176591 exit 1 sudo tee /dev/pts/2 > /dev/null
```

```
[& krafi@pop-os ]--[⌚ 04:15 PM]--> www.krafi.info
└─ Documents/arciture rudn/14
-> ./semaphore.sh 10 5
```

Процесс 176684: Ресурс получен! Использую в течение 5 сек.  
Процесс 176684: Ресурс освобожден.

```
[& krafi@pop-os ]--[⌚ 04:16 PM]--> www.krafi.info
└─ Documents/arciture rudn/14 took 5s
```

### **Terminal 2 (/dev/pts/2):**

Процесс 1234: Ресурс получен! Использую в течение 5 сек.  
Процесс 1235: Ресурс занят. Ожидание... (0 сек.)  
Процесс 1235: Ресурс занят. Ожидание... (1 сек.)  
...  
Процесс 1235: Ресурс занят. Ожидание... (4 сек.)  
Процесс 1234: Ресурс освобожден.  
Процесс 1235: Ресурс получен! Использую в течение 5 сек.  
Процесс 1235: Ресурс освобожден.

### **Доработка для взаимодействия трех и более процессов:**

Предложенный скрипт уже поддерживает неограниченное количество процессов, так как механизм проверки файла (-f "\$semaphore\_file") атомарен. Все процессы будут ждать в цикле, пока процесс, владеющий ресурсом, не удалит файл. Порядок получения ресурса после освобождения не гарантируется (решается очередью планировщика ОС).

### **Задание 2: Реализация команды man**

#### **Листинг скрипта myman.sh:**

```
#!/bin/bash

# Проверяем, передан ли аргумент
if [ $# -eq 0 ]; then
    echo "Ошибка: Укажите имя команды для просмотра справки."
    echo "Пример: ./myman.sh ls"
    exit 1
fi

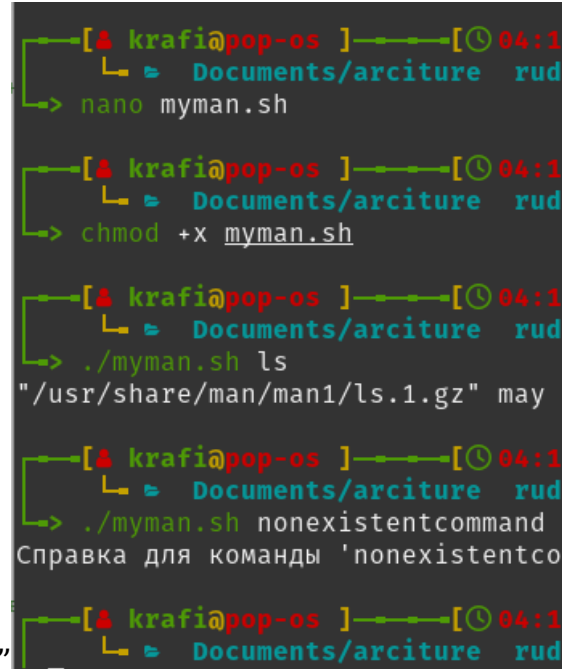
command_name=$1
man_page_path="/usr/share/man/man1/${command_name}.1.gz"

# Проверяем существование файла справки
if [ -f "$man_page_path" ]; then
    # Если файл найден, открываем его с помощью less
    less "$man_page_path"
elif [ -f "${man_page_path%.gz}" ]; then
    # На случай, если файл не запакован (без .gz)
```

```
less "${man_page_path%.gz}"
else
echo "Справка для команды '$command_name' не найдена в
/usr/share/man/man1/."
exit 1
fi
```

### Результат выполнения:

```
chmod +x myman.sh
./myman.sh ls # Выведет мануал для 'ls'
./myman.sh nonexistentcommand # Выведет: "Справка для
```



```
[krafi@pop-os ~]$ nano myman.sh
[krafi@pop-os ~]$ chmod +x myman.sh
[krafi@pop-os ~]$ ./myman.sh ls
"/usr/share/man/man1/ls.1.gz" may
[krafi@pop-os ~]$ ./myman.sh nonexistentcommand
Справка для команды 'nonexistentco
команды 'nonexistentcommand' не найдена..."
```

### Задание 3: Генерация случайной последовательности букв

#### Листинг скрипта random\_letters.sh:

```
#!/bin/bash

# Функция для генерации одной случайной буквы
get_random_letter() {
# Генерируем случайное число от 0 до 25 (26 букв)
local random_num=$(( RANDOM % 26 ))
```

```
# Преобразуем число в букву. 65 - код 'A' в ASCII, 97 - код 'a'
# Для заглавных: printf "\x$(printf %x $((65 + random_num)))"
printf "\x$(printf %x $((97 + random_num)))"
}
```

```
# Задаем длину последовательности (можно вынести в
аргумент)
length=10
result=""
```

```
# Цикл для генерации последовательности заданной длины
for (( i=0; i<$length; i++ )); do
# Конкатенируем буквы в строку
result="${result}${get_random_letter}"
done
```

```
echo "Случайная последовательность: $result"
```

**Альтернативная, более эффективная реализация с помощью массива:**

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

```
length=10
declare -a letters
```

```
for (( i=0; i<$length; i++ )); do
random_num=$(( RANDOM % 26 ))
# Заполняем массив кодами букв
letters[i]=$((97 + random_num))
done
```

```
# Преобразуем массив кодов символов в реальную строку
printf -v result "\x$(printf %x "${letters[@]}")"
echo "Случайная последовательность: $result"
```

**Результат выполнения:**

```
chmod +x random_letters.sh
./random_letters.sh
# Случайная последовательность: qzxtfgpvon
./random_letters.sh
```

```

[& kraf@pop-os ]-----[⌚]
└─ Documents/arciture
-> nano random_letters.sh

[& kraf@pop-os ]-----[⌚]
└─ Documents/arciture
-> sudo chmod +x ./random_let

[& kraf@pop-os ]-----[⌚]
└─ Documents/arciture
-> ./random_letters.sh
Случайная последовательность:

[& kraf@pop-os ]-----[⌚]
└─ Documents/arciture
-> ./random_letters.sh
Случайная последовательность:

[& kraf@pop-os ]-----[⌚]
└─ Documents/arciture
->

```

# Случайная последовательность: bkasejldm

## Ответы на контрольные вопросы

4. **Найдите синтаксическую ошибку в следующей строке:** `while [$1 != "exit"]` **Ошибка:** Недостаточно пробелов. Конструкция `[` — это actually команда (синоним `test`), поэтому она требует пробелов вокруг себя и вокруг операторов. **Правильный вариант:** `while [ "$1" != "exit" ]`
5. **Как объединить (конкатенация) несколько строк в одну?**
  - **В переменную:** `new_string="$string1$string2"`
  - **В выводе:** `echo "$string1" "$string2"`
  - **С разделителем:** `joined_string=$(printf "%s\n" "${array[@]}")` для массива.
6. **Найдите информацию об утилите `seq`.**



**Какими иными способами можно реализовать её функционал? seq** генерирует последовательность чисел. **Аналоги на bash:**

- Цикл с C-подобным синтаксисом: `for ((i=1; i<=5; i++)); do ... done`
- Фигурные скобки: `echo {1..5}` (но это не итерация, а генерация списка).
- `while`-цикл с инкрементом счетчика.

7. **Какой результат даст вычисление выражения  $\$(10/3)$ ?** Это вызовет ошибку, так как `$(...)` — это подстановка вывода команды. Команды `10/3` не существует. Для арифметических вычислений нужно использовать `$((10/3))` (вернет 3).

8. **Укажите кратко основные отличия командной оболочки `zsh` от `bash`.**

- **Zsh:** Более развитые возможности автодополнения (более умное и настраиваемое), смена каталога по имени без `cd`, глобальные псевдонимы, улучшенная обработка подстановки путей (globbing), темы для приглашения командной строки.
- **Bash:** Стандартная оболочка по умолчанию в большинстве дистрибутивов, более широко распространена, скрипты на `bash` более переносимы. `Zsh` часто используют как интерактивную оболочку, а `bash` — для системных скриптов.

9. **Проверьте, верен ли синтаксис данной конструкции `for ((a=1; a <= LIMIT; a++))`** Да, синтаксис верен, если переменная `LIMIT` предварительно определена. Это корректный C-подобный синтаксис для арифметического цикла в `bash`.

10. **Сравните язык `bash` с какими-либо языками программирования.**

- **Преимущества bash:**
  - Идеален для автоматизации вызова системных утилит (намного проще, чем на C/Python).
  - Не требует компиляции, удобен для быстрого прототипирования.
  - Встроенная работа с потоками данных (pipes, перенаправления).
- **Недостатки bash:**
  - Медленнее компилируемых языков (C, Go).
  - Неудобен для сложных структур данных (массивы, объекты) и алгоритмов.
  - Чувствителен к пробелам и специальным символам, что может приводить к ошибкам.
  - Отсутствие строгой типизации.