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

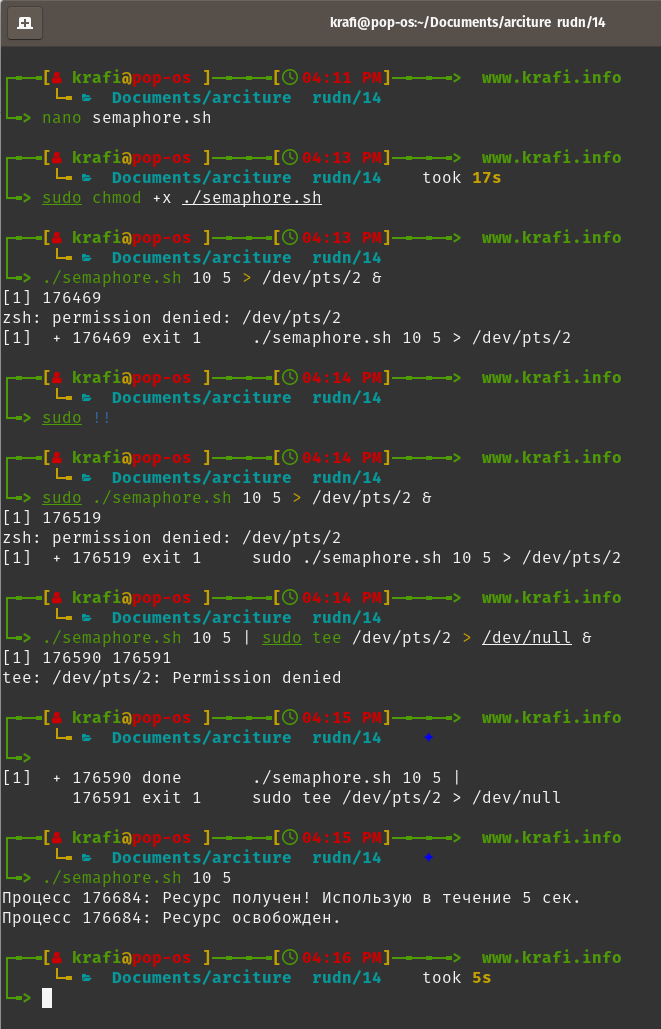
**Инструкция по запуску и скриншот:**

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.*

1. **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.*



**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.

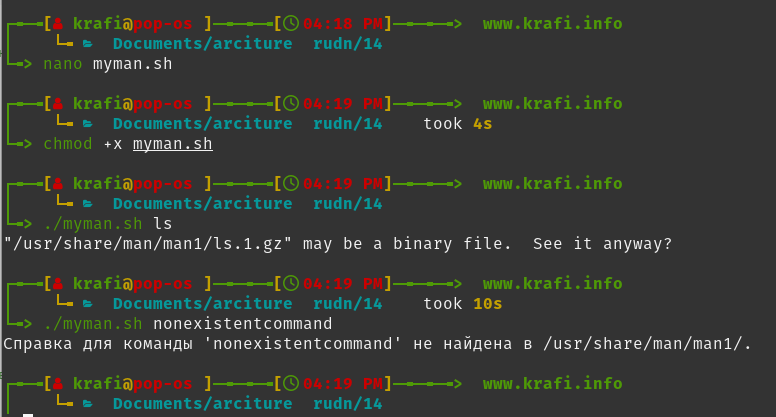
**Доработка для взаимодействия трех и более процессов:** Предложенный скрипт уже поддерживает неограниченное количество процессов, так как механизм проверки файла (-f "$semaphore\_file") атомарен. Все процессы будут ждать в цикле, пока процесс, владеющий ресурсом, не удалит файл. Порядок получения ресурса 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 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  
# Случайная последовательность: bkacejildm

### **Ответы на контрольные вопросы**

1. **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 [ "$1" != "exit" ]
2. **Как объединить (конкатенация) несколько строк в одну?**
   * **В переменную:** new\_string="$string1$string2"
   * **В выводе:** echo "$string1" "$string2"
   * **С разделителем:** joined\_string=$(printf "%s\n" "${array[@]}") для массива.
3. **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.
4. **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).
5. **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.
6. **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.
7. **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.