

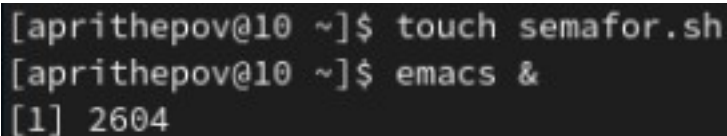
12 Лабораторная Работа

Прищепов Александр НПМ-03-21

Введение:

- Цель работы: Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.
Ход Работы:
- 1. Напишем командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом).

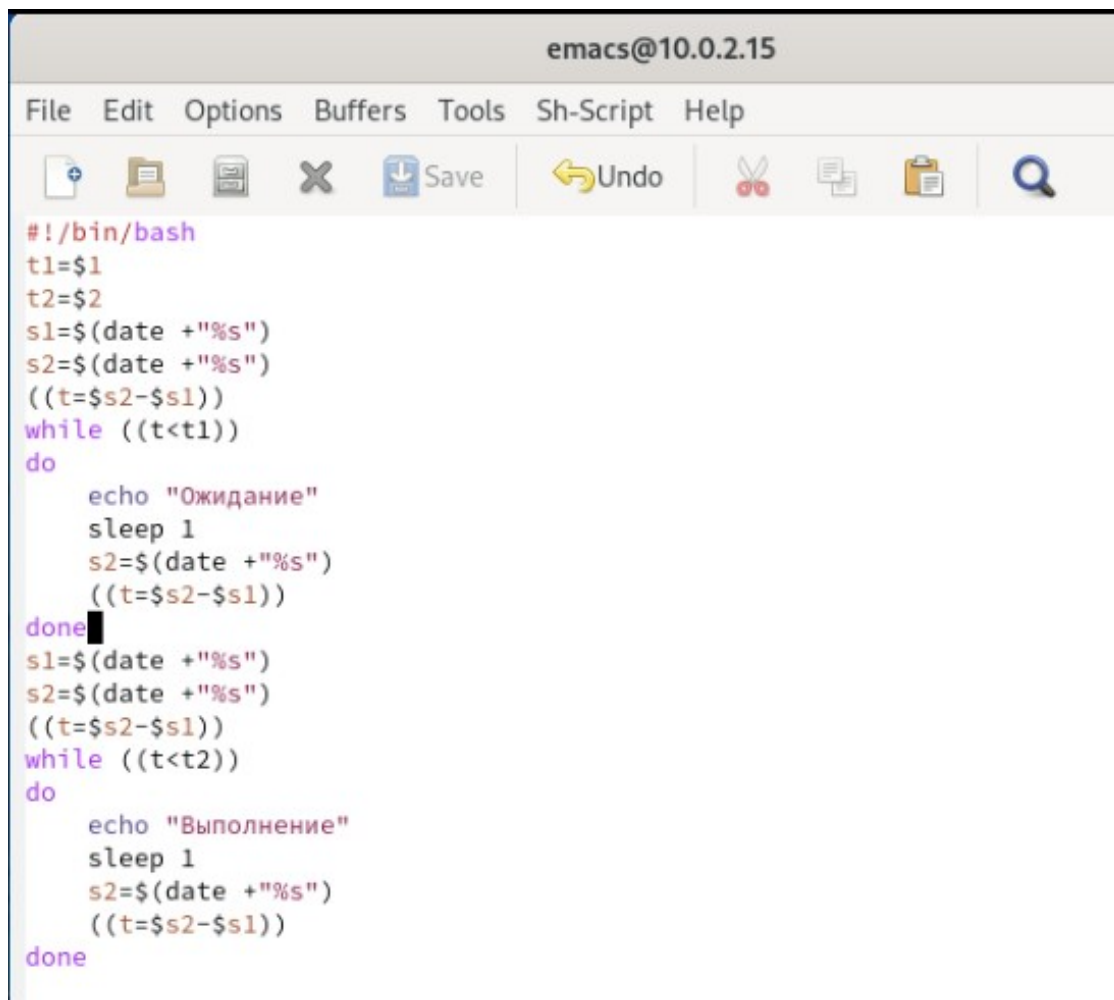
Для выполнения данной задачи создадим файл `semafor.sh` и откроем его в `emacs` (рис.1).



```
[aprithepov@10 ~]$ touch semafor.sh
[aprithepov@10 ~]$ emacs &
[1] 2604
```

изображение

В файле напишем соответствующий скрипт (рис.2) и проверим его работу (команда `./semafor.sh 2 4`), предварительно добавив права на выполнение (команда `chmod +x semafor.sh`) (рис.3).



```
emacs@10.0.2.15
File Edit Options Buffers Tools Sh-Script Help
[Icons: New, Open, Save, Close, Save As, Undo, Cut, Copy, Paste, Find]

#!/bin/bash
t1=$1
t2=$2
s1=$(date +%s)
s2=$(date +%s)
((t=s2-s1))
while ((t<t1))
do
    echo "Ожидание"
    sleep 1
    s2=$(date +%s)
    ((t=s2-s1))
done
s1=$(date +%s)
s2=$(date +%s)
((t=s2-s1))
while ((t<t2))
do
    echo "Выполнение"
    sleep 1
    s2=$(date +%s)
    ((t=s2-s1))
done
```

изображение

```
[aprithepov@10 ~]$ chmod +x semafor.sh
[aprithepov@10 ~]$ ./semafor.sh 2 4
./semafor.sh: строка 6: синтаксическая ошибка рядом с неожиданным маркером «(»
./semafor.sh: строка 6: `((t=$s2-$s1))'
[aprithepov@10 ~]$ chmod +x semafor.sh
[aprithepov@10 ~]$ ./semafor.sh 2 4
./semafor.sh: строка 6: синтаксическая ошибка рядом с неожиданным маркером «(»
./semafor.sh: строка 6: `((t=$s2-$s1))'
[aprithepov@10 ~]$ ./semafor.sh 2 4
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
```

изображение

Затем изменим скрипт так, чтобы можно было запускать командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (рис.4).

File Edit Options Buffers Tools Sh-Script Help



```
#!/bin/bash
function ozhidanie
{
    s1=$(date +%s)
    s2=$(date +%s)
    ((t=s2-s1))
    while ((t<t1))
    do
        echo "Ожидание"
        sleep 1
        s2=$(date +%s)
        ((t=s2-s1))
    done
}
function vipolnenie
{
    s1=$(date +%s)
    s2=$(date +%s)
    ((t=s2-s1))
    while ((t<t2))
    do
        echo "Выполнение"
        sleep 1
        s2=$(date +%s)
        ((t=s2-s1))
    done
}
t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Выход" ]
    then ozhidanie
    fi
    if [ "$command" == "Выполнение" ]
    then vipolnenie
    fi
    echo "Следующее действие: "
    read command
done
```

U:~-- semafor.sh All L36 (Shell-script[sh])

изображение

Проверим его работу (например, команда `./semafor.sh 2 4 Ожидание > /dev/pts/1`) и увидим, что нам отказано в доступе (рис.6). Но при этом скрипт работает корректно (рис.7) при вводе команды `./semafor.sh 2 4 Ожидание`.

```
[aprithepov@10 ~]$ ./semafor.sh 2 4 Ожидание > /dev/pts/1 &
[2] 3147
bash: /dev/pts/1: Отказано в доступе
[2]+  Выход 1                  ./semafor.sh 2 4 Ожидание > /dev/pts/1
[aprithepov@10 ~]$ ./semafor.sh 2 4 Ожидание > /dev/pts/2 &
[2] 3166
bash: /dev/pts/2: Отказано в доступе
[2]+  Выход 1                  ./semafor.sh 2 4 Ожидание > /dev/pts/2
[aprithepov@10 ~]$ ./semafor.sh 2 4 Выполнение > /dev/pts/1 &
[2] 3171
bash: /dev/pts/1: Отказано в доступе
[2]+  Выход 1                  ./semafor.sh 2 4 Выполнение > /dev/pts/1
[aprithepov@10 ~]$ ./semafor.sh 2 4 Выполнение > /dev/pts/2 &
[2] 3176
bash: /dev/pts/2: Отказано в доступе
[2]+  Выход 1                  ./semafor.sh 2 4 Выполнение > /dev/pts/2
[aprithepov@10 ~]$
```

изображение

```
[aprithepov@10 ~]$ ./semafor.sh 2 4 Ожидание
Следующее действие:
Ожидание
Следующее действие:
Выполнение
Выполнение
Выполнение
Выполнение
Следующее действие:
Выход
Выход
[aprithepov@10 ~]$
```

изображение

3. Используя встроенную переменную `$RANDOM`, напомним командный файл, генерирующий случайную последовательность букв

латинского алфавита. Для этого создадим файл random.sh и откроем его в emacs.

Напишем скрипт для выполнения 3 задания (рис.14).

```
#!/bin/bash
a=$1
for ((i=0; i<$a; i++))
do
    ((char=$((RANDOM%26+1)))
    case $char in
        1) echo -n a;; 2) echo -n b;; 3) echo -n c;; 4) echo -n d;; 5) echo -n e;; 6) echo -n f;; 7) echo -n g;; 8) echo -n h;; 9) echo -n i;; 10) echo -n j;; 11) echo -n k;;
        12) echo -n l;; 13) echo -n m;; 14) echo -n n;; 15) echo -n o;; 16) echo -n p;; 17) echo -n q;; 18) echo -n r;; 19) echo -n s;; 20) echo -n t;; 21) echo -n u;; 22) echo -n v;;
        23) echo -n w;; 24) echo -n x;; 25) echo -n y;; 26) echo -n z;;
    esac
done
echo
```

изображение

Проверим его работу (команда ./random.sh 158), предварительно дав ему право на выполнение с помощью команды chmod +x random.sh (рис.15).

```
[aprithepov@10 ~]$ chmod +x random.sh
[aprithepov@10 ~]$ ./random.sh 158
tbiefihlynpphpxixvedkpkppbohndgehzffkjigvenazaosuhjrrjawysshzemjmkjcrneqbpjukwia
bflmzpulpdytwejvqiayasyabthgmtximnurmcoejconzpnqookuvsxmvzarvvpaafrwbwoprsp
[aprithepov@10 ~]$
```

изображение

Выводы

Я изучил основы программирования в оболочке ОС UNIX и научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Контрольные вопросы

1). while [\$1 != "exit"]

В данной строчке допущены следующие ошибки:

- не хватает пробелов после первой скобки [и перед второй скобкой]
- выражение \$1 необходимо взять в "", потому что эта переменная может содержать пробелы.

Таким образом, правильный вариант должен выглядеть так: while ["\$1" != "exit"]

2). Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами:

- Первый: VAR1="Hello,
"VAR2=" World"

```
VAR3="V A R 1VAR2"
```

```
echo "$VAR3"
```

Результат: Hello, World

- Второй: VAR1="Hello,"

```
VAR1+=" World"
```

```
echo "$VAR1"
```

Результат: Hello, World

3). Команда `seq` в Linux используется для генерации чисел от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT.

Параметры:

- `seq LAST`: если задан только один аргумент, он создает числа от 1 до LAST с шагом шага, равным 1. Если LAST меньше 1, значение не выдает.
- `seq FIRST LAST`: когда заданы два аргумента, он генерирует числа от FIRST до LAST с шагом 1, равным 1. Если LAST меньше FIRST, он не выдает никаких выходных данных.
- `seq FIRST INCREMENT LAST`: когда заданы три аргумента, он генерирует числа от FIRST до LAST на шаге INCREMENT. Если LAST меньше, чем FIRST, он не производит вывод.
- `seq -f «FORMAT» FIRST INCREMENT LAST`: эта команда используется для генерации последовательности в форматированном виде. FIRST и INCREMENT являются необязательными.
- `seq -s «STRING» ПЕРВЫЙ ВКЛЮЧЕНО`: Эта команда используется для STRING для разделения чисел. По умолчанию это значение равно /n. FIRST и INCREMENT являются необязательными.
- `seq -w FIRST INCREMENT LAST`: эта команда используется для выравнивания ширины путем заполнения начальными нулями. FIRST и INCREMENT являются необязательными.

4). Результатом данного выражения $((10/3))$ будет 3, потому что это целочисленное деление без остатка.

5). Отличия командной оболочки `zsh` от `bash`:

- В `zsh` более быстрое автодополнение для `cd` с помощью `Tab`
- В `zsh` существует калькулятор `zcalc`, способный выполнять вычисления внутри терминала

- В zsh поддерживаются числа с плавающей запятой
- В zsh поддерживаются структуры данных «хэш»
- В zsh поддерживается раскрытие полного пути на основе неполных данных
- В zsh поддерживается замена части пути
- В zsh есть возможность отображать разделенный экран, такой же как разделенный экран vim

6). `for((a=1; a<= LIMIT; a++))` синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать \$ перед переменными ().

7). Преимущества скриптового языка bash:

- Один из самых распространенных и ставится по умолчанию в большинстве дистрибутивах Linux, MacOS
- Удобное перенаправление ввода/вывода
- Большое количество команд для работы с файловыми системами Linux
- Можно писать собственные скрипты, упрощающие работу в Linux

Недостатки скриптового языка bash:

- Дополнительные библиотеки других языков позволяют выполнить больше действий
- Bash не является языком общего назначения
- Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на скорости выполнения этого скрипта
- Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий.