

Титульный лист

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Лабораторная работа 13

По дисциплине "Операционные системы"

Выполнил:

Студент группы НПМбв-01-19

Студенческий билет №: 1032187017

Кушнирчук Дарья Вадимовна

Руководитель: Валиева Татьяна Рефатовна

Москва 2023

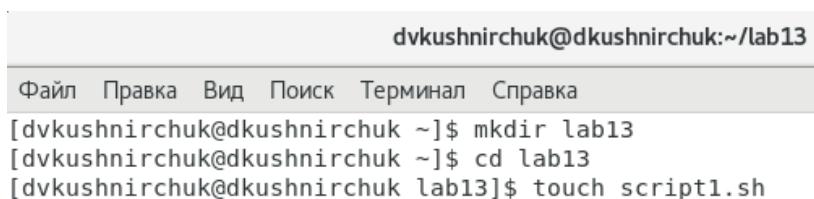
Цель работы

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

Начало 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). Запустить командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой ($>/dev/tty\#$, где $\#$ — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.

- Для этого создадим файл `script1.sh`



```
dvkushnirchuk@dkushnirchuk:~/lab13
Файл  Правка  Вид  Поиск  Терминал  Справка
[dvkushnirchuk@dkushnirchuk ~]$ mkdir lab13
[dvkushnirchuk@dkushnirchuk ~]$ cd lab13
[dvkushnirchuk@dkushnirchuk lab13]$ touch script1.sh
```

Рисунок 1

- Откроем файл с помощью команды `vi` и запишем следующий код.

```
dvkushnirchuk@dkushnirchuk:~/lab13
Файл Правка Вид Поиск Терминал Справка
t1=$1
t2=$2
s1=$(date +%s)
s2=$(date +%s)
((t=$s2-$s1))
while ((t < t1))
do
    echo "Ожидание разблокировки файла"
    sleep 1
    s2=$(date +%s)
    ((t=$s2-$s1))
    touch lockfile
done
s1=$(date +%s)
s2=$(date +%s)
((t=$s2-$s1))
while ((t < t2))
do
    echo "Открытие"
    sleep 1
    s2=$(date +%s)
    ((t=$s2-$s1))
done
rm lockfile
echo "Готово"
~
```

Рисунок 2

- Предоставим права владельцу и проверим работу скрипта

```
dvkushnirchuk@dkushnirchuk:~/lab13
Файл Правка Вид Поиск Терминал Справка
[dvkushnirchuk@dkushnirchuk ~]$ cd lab13
[dvkushnirchuk@dkushnirchuk lab13]$ vi script1.sh
[dvkushnirchuk@dkushnirchuk lab13]$ vi script1.sh
[dvkushnirchuk@dkushnirchuk lab13]$ chmod +x script1.sh
[dvkushnirchuk@dkushnirchuk lab13]$ ./script1.sh 3 4
Ожидание разблокировки файла
Ожидание разблокировки файла
Ожидание разблокировки файла
Открытие
Открытие
Открытие
Открытие
Готово
[dvkushnirchuk@dkushnirchuk lab13]$
```

Рисунок 3

2. Напишем командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 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 данной задачи я создал файл: semaphore.sh (Рисунки 1,2) и написал соответствующий скрипт.

- Сперва изучим содержимое каталога `/usr/share/man/man1`

```
dvkushnirchuk@dkushnirchuk:/usr/share/man/man1
Файл  Правка  Вид  Поиск  Терминал  Справка
[dvkushnirchuk@dkushnirchuk ~]$ cd /usr/share/man/man1
[dvkushnirchuk@dkushnirchuk man1]$ ls
.: .1.gz
[.1.gz
a2p.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
abrt-action-analyze-core.1.gz
abrt-action-analyze-oops.1.gz
abrt-action-analyze-python.1.gz
abrt-action-analyze-vmcore.1.gz
abrt-action-analyze-vulnerability.1.gz
abrt-action-analyze-xorg.1.gz
abrt-action-check-oops-for-hw-error.1.gz
abrt-action-generate-backtrace.1.gz
```

Рисунок 4

- Далее создадим файл `script2.sh`

```
[dvkushnirchuk@dkushnirchuk lab13]$ touch script2.sh
[dvkushnirchuk@dkushnirchuk lab13]$ vi script2.sh
```

Рисунок 5

- При помощи команды `vi` откроем файл и запишем следующий код.

```
dvkushnirchuk@dkushnirchuk:~
Файл  Правка  Вид  Поиск  Терминал  Справка
#!/bin/bash
c=$1
if [ -f /usr/share/man/man1/${c}.1.gz ]
then
    gunzip -c /usr/share/man/man1/${c}.1.gz | less
else
    echo "There is no help for this command"
fi
```

Рисунок 6

- Предоставим права владельцу и проверим работу скрипта

```
[dvkushnirchuk@dkushnirchuk lab13]$ chmod +x script2.sh
[dvkushnirchuk@dkushnirchuk lab13]$ ./script2.sh touch
[dvkushnirchuk@dkushnirchuk lab13]$ ./script2.sh mkdir
```

Рисунок 7

Рисунок 8

```
Файл  Правка  Вид  Поиск  Терминал  Справка
.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.43.3.
.TH MKDIR "1" "November 2020" "GNU coreutils 8.22" "User Commands"
.SH NAME
mkdir \- make directories
.SH SYNOPSIS
.B mkdir
[\fIOPTION\fR]... [\fIDIRECTORY\fR]...
.SH DESCRIPTION
.\" Add any additional description here
.PP
Create the DIRECTORY(ies), if they do not already exist.
.PP
Mandatory arguments to long options are mandatory for short options too.
.TP
\fB\m\fR, \fB\--mode\fR=\fIMODE\fR
set file mode (as in chmod), not a=rwx \- umask
.TP
\fB\p\fR, \fB\--parents\fR
no error if existing, make parent directories as needed
.TP
\fB\v\fR, \fB\--verbose\fR
print a message for each created directory
.TP
\fB\Z\fR
set SELinux security context of each created directory
to the default type
.TP
.
```

Рисунок 9

3. Используя встроенную переменную \$RANDOM, напомним командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтем, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

- Для этого создадим файл script3.sh

```
[dvkushnirchuk@dkushnirchuk lab13]$ touch script3.sh
```

Рисунок 10

- Откроем файл с помощью команды vi и запишем следующий код.

```
dvkushnirchuk@dk
Файл  Правка  Вид  Поиск  Терминал  Справка
#!/bin/bash
k=$1
declare -a rand
rand=({a..z})
let limit=25
let i=k+1
while ((i-=1))
do
    numb=$RANDOM
    let numb%=limit
    output=$output${rand[numb]}
done
echo $output
```

Рисунок 11

- Предоставим права владельцу и проверим работу скрипта

```
[dvkushnirchuk@dkushnirchuk lab13]$ chmod +x script3.sh
[dvkushnirchuk@dkushnirchuk lab13]$ ./script3.sh 1

[dvkushnirchuk@dkushnirchuk lab13]$ ./script3.sh 2
`
[dvkushnirchuk@dkushnirchuk lab13]$ ./script3.sh 10
^^`
[dvkushnirchuk@dkushnirchuk lab13]$ ./script3.sh 77
Z]aa`Z^`Z[`aZZa_][a[]_
[dvkushnirchuk@dkushnirchuk lab13]$ vi script3.sh
[dvkushnirchuk@dkushnirchuk lab13]$ ./script3.sh 1
q
[dvkushnirchuk@dkushnirchuk lab13]$ ./script3.sh 5
nutra
[dvkushnirchuk@dkushnirchuk lab13]$ ./script3.sh 27
lklhqlnyywdidmbmdwhmturrbmt
[dvkushnirchuk@dkushnirchuk lab13]$ █
```

Рисунок 12

Вывод

Мы изучили основы программирования в оболочке ОС UNIX. Научились 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="\$VAR1\$VAR2" 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:

- `eq LAST`: если задан только один аргумент, он создает числа от 1 до LAST с шагом шага, равным 1. Если LAST меньше 1, значение `is` 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 `$((10/3))` будет 3, потому что это целочисленное деление без остатка.
5. Отличия командной оболочки `zsh` от `bash`:
- В `zsh` более быстрое автодополнение для `cd` с помощью `Tab`
 - В `zsh` существует калькулятор `zcalc`, способный выполнять вычисления внутри терминала
 - В `zsh` поддерживаются числа с плавающей запятой
 - В `zsh` поддерживаются структуры данных «хэш»
 - В `zsh` поддерживается раскрытие полного пути на основе неполных данных
 - В `zsh` поддерживается замена части пути
 - В `zsh` есть возможность отображать разделенный экран, такой же как разделенный экран `vim`
6. `or ((a=1; a <= LIMIT; a++))` синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать `$` 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