

Отчёт по лабораторной работе № 12

дисциплина: Операционные системы

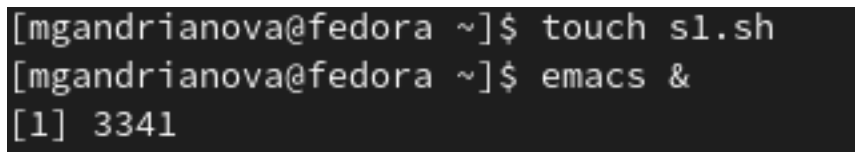
Андрианова Марина Георгиевна

Цель работы

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

Выполнение лабораторной работы

1. Создала файл `s1.sh` (рис.1) и написала соответствующий скрипт (рис.2): 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).



```
[mgandrianova@fedora ~]$ touch s1.sh
[mgandrianova@fedora ~]$ emacs &
[1] 3341
```

Рис.1: Создание файла `s1.sh`

```
#!/bin/bash
t1=$1 #Время ожидания
t2=$2 #Время выполнения
s1=$(date +%s) #Счётчик времени s1 (сек)
s2=$(date +%s) #Счётчик времени s2 (сек)
((t=$s2-$s1)) #Счётчик времени t (сек). Будет изменяться в цикле
while ((t<t1)) #Цикл ожидания
do
    echo "Ожидание" #Вывод сообщения
    sleep 1 #Пауза на 1 сек для изменения счётчика
    s2=$(date +%s)
    ((t=$s2-$s1))
done
s1=$(date +%s) #Обновление счётчика s1
s2=$(date +%s) #Обновление счётчика s1
((t=$s2-$s1)) #Обновление счётчика t
while ((t<t2)) #Цикл выполнения
do
    echo "Выполнение" #Вывод сообщения
    sleep 1 #Пауза на 1 сек для изменения счётчика
    s2=$(date +%s)
    ((t=$s2-$s1))
done
```

U:***- s1.sh All L24 (Shell-script[sh])

Рис.2: Пишем командный файл s1.sh

Проверяем работу написанного скрипта (команда “./s1.sh 2 5”), предварительно добавив право на исполнение файла(команда “chmod +x s1.sh”). Скрипт работает корректно(рис.3).

```
[mgandrianova@fedora ~]$ chmod +x s1.sh
[1]+  Завершён      emacs
[mgandrianova@fedora ~]$ ./s1.sh 2 5
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
[mgandrianova@fedora ~]$
```

Рис.3: Проверка скрипта

После этого изменяем скрипт так, чтобы его можно было выполнять в нескольких терминалах(рис.4). Проверила его работу(команда “./s1.sh 3 5 Ожидание > /dev/pts/2 &” и команда “./s1.sh 3 5 Ожидание > /dev/tty2”). При этом ни одна из команд не сработала, выводя сообщение “Отказано в доступе”. При этом скрипт работает корректно(рис.5).

```
#!/bin/bash
function o
{
    s1=$(date +%s) #Счётчик времени s1 (сек)
    s2=$(date +%s) #Счётчик времени s2 (сек)
    ((t=s2-s1)) #Счётчик времени t (сек). Будет изменяться в цикле
    while ((t<t1)) #Цикл ожидания
    do
        echo "Ожидание" #Вывод сообщения
        sleep 1 #Пауза на 1 сек для изменения счётчика
        s2=$(date +%s)
        ((t=s2-s1))
    done
}
function v
{
    s1=$(date +%s) #Обновление счётчика s1
    s2=$(date +%s) #Обновление счётчика s1
    ((t=s2-s1)) #Обновление счётчика t
    while ((t<t2)) #Цикл выполнения
    do
        echo "Выполнение" #Вывод сообщения
        sleep 1 #Пауза на 1 сек для изменения счётчика
        s2=$(date +%s)
        ((t=s2-s1))
    done
}
t1=$1 #Время ожидания
t2=$2 #Время выполнения
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then o
    fi
    if [ "$command" == "Выполнение" ]
    then v
    fi
    echo "Следующее действие: "
    read command
done
```

Рис.4: Изменение командного файла s1.sh

```
[mgandrianova@fedora ~]$ chmod +x sl.sh
[1]+  Завершён          emacs
[mgandrianova@fedora ~]$ ./sl.sh 3 5 Ожидание > /dev/pts/2 &
[1] 4032
bash: /dev/pts/2: Отказано в доступе
[1]+  Выход 1          ./sl.sh 3 5 Ожидание > /dev/pts/2
[mgandrianova@fedora ~]$ ./sl.sh 3 5 Ожидание > /dev/pts/1 &
[1] 4041
bash: /dev/pts/1: Отказано в доступе
[1]+  Выход 1          ./sl.sh 3 5 Ожидание > /dev/pts/1
[mgandrianova@fedora ~]$ ./sl.sh 3 5 Ожидание > /dev/tty2
```

Рис.5: Проверка скрипта

2. Реализовала команду `man` с помощью командного файла. Изучила содержимое каталога `/usr/share/man/man1`: сначала перешла в него командой `cd /usr/share/man/man1`, а затем вывела его содержимое (команда `ls`) (рис.6). В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой `less` сразу же просмотрев содержимое справки. Командный файл должен получать в виде аргумента командной строки название команды и в виде результата выдавать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге `man1`.

```
[mgandrianova@fedora ~]$ cd /usr/share/man/man1
[mgandrianova@fedora man1]$ ls
:.1.gz
' [.1.gz'
ab.1.gz
abrt.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-java.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-find-bodhi-update.1.gz
abrt-action-generate-backtrace.1.gz
abrt-action-generate-core-backtrace.1.gz
abrt-action-install-debuginfo.1.gz
abrt-action-list-dsos.1.gz
abrt-action-notify.1.gz
abrt-action-perform-ccpp-analysis.1.gz
abrt-action-save-package-data.1.gz
abrt-action-trim-files.1.gz
abrt-applet.1.gz
abrt-auto-reporting.1.gz
abrt-bodhi.1.gz
abrt-cli.1.gz
abrt-dump-journal-core.1.gz
```

Рис.6: Переход в каталог и вывод его содержимого

Для данной задачи я создала файл s2.sh(рис.7) и написала соответствующий скрипт(рис.8).

```
[mgandrianova@fedora man1]$ cd
[mgandrianova@fedora ~]$ touch s2.sh
[mgandrianova@fedora ~]$ emacs &
[1] 4185
```

Рис.7: Создание файла s2.sh

```
#!/bin/bash
c=$1 #Инициализация названия команды
if [ -f /usr/share/man/man1/$c.1.gz ] #Проверка существования справки
then
    gunzip -c /usr/share/man/man1/$1.1.gz | less #Распаковка архива со справкой(если она есть)
else
    echo "Справки по данной команде нет"
fi
```

U:--- s2.sh All L9 (Shell-script[sh])
Wrote /home/mgandrianova/s2.sh

Рис.8: Пишем командный файл s2.sh

Проверяем работу написанного скрипта (команды “./s2.sh touch” и “./s2.sh rm”), предварительно добавив право на исполнение файла(команда “chmod +x s2.sh”)(рис.9). Скрипт работает корректно(рис.10,рис.11).

```
[mgandrianova@fedora ~]$ chmod +x s2.sh
[1]+  Завершён      emacs
[mgandrianova@fedora ~]$ ./s2.sh touch
[mgandrianova@fedora ~]$ ./s2.sh rm
```

Рис.9: Предоставление права на исполнение и проверка скрипта


```
mgandrianova@fedora:~ — /bin/bash ./s2.sh touch

.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH TOUCH "1" "March 2022" "GNU coreutils 8.32" "User Commands"
.SH NAME
touch \- change file timestamps
.SH SYNOPSIS
.B touch
[\\fI\\,OPTION\\/\\fR]... \\fI\\,FILE\\/\\fR...
.SH DESCRIPTION
.\" Add any additional description here
.PP
Update the access and modification times of each FILE to the current time.
.PP
A FILE argument that does not exist is created empty, unless \\fB\\-c\\fR or \\fB\\-h\\fR
is supplied.
.PP
A FILE argument string of \\- is handled specially and causes touch to
change the times of the file associated with standard output.
.PP
Mandatory arguments to long options are mandatory for short options too.
.TP
\\fB\\-a\\fR
change only the access time
.TP
\\fB\\-c\\fR, \\fB\\-\\-no\\-create\\fR
do not create any files
.TP
\\fB\\-d\\fR, \\fB\\-\\-date\\fR=\\fI\\,STRING\\/\\fR
parse STRING and use it instead of current time
.TP
\\fB\\-f\\fR
:

```

Рис.10: Результат после введения команды “./s2.sh touch”

```
mgandrianova@fedora:~ — /bin/bash ./s2.sh rm
.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH RM "1" "March 2022" "GNU coreutils 8.32" "User Commands"
.SH NAME
rm \- remove files or directories
.SH SYNOPSIS
.B rm
[ \fI\,OPTION\ \fR ]... [ \fI\,FILE\ \fR ]...
.SH DESCRIPTION
This manual page
documents the GNU version of
.BR rm .
.B rm
removes each specified file. By default, it does not remove
directories.
.P
If the \fI\-I\fR or \fI\--interactive=once\fR option is given,
and there are more than three files or the \fI\-r\fR, \fI\-R\fR,
or \fI\--recursive\fR are given, then
.B rm
prompts the user for whether to proceed with the entire operation. If
the response is not affirmative, the entire command is aborted.
.P
Otherwise, if a file is unwritable, standard input is a terminal, and
the \fI\-f\fR or \fI\--force\fR option is not given, or the
\fI\-i\fR or \fI\--interactive=always\fR option is given,
.B rm
prompts the user for whether to remove the file. If the response is
not affirmative, the file is skipped.
.SH OPTIONS
.PP
:[]
```

Рис.11: Результат после введения команды “./s2.sh rm”

3. Создала файл s3.sh(рис.12) и написала соответствующий скрипт(рис.13). Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтём, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767. Добавила право на исполнение файла(команда “chmod +x s3.sh”)(рис.14). Скрипт работает корректно(рис.14).

```
[mgandrianova@fedora ~]$ touch s3.sh
[mgandrianova@fedora ~]$ emacs &
[1] 4453
```

Рис.12: Создание файла s3.sh

```
#!/bin/bash
c=$1 #Инициализация количества символов
for (( i=0; i<$c; i++)) #Цикл вывода нужного количества символов
do
    (( char=$RANDOM%26+1 )) #Случайные номера от 1 до 26
    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
```

U:--- s3.sh All L15 (Shell-script[sh])
Wrote /home/mgandrianova/s3.sh

Рис.13: Пишем командный файл s3.sh

```
[mgandrianova@fedora ~]$ chmod +x s3.sh
[1]+  Завершён      emacs
[mgandrianova@fedora ~]$ ./s3.sh 5
dsxxkx
[mgandrianova@fedora ~]$ ./s3.sh 5
tkuio
[mgandrianova@fedora ~]$ ./s3.sh 100
slzurshlbrkwhwsmtjcodfaqdwcnujhvoqrisgcvuafdkxawfdoomypnmzhkassiidjxqolbrxulcgpueyraymddbdtraqvfw
[mgandrianova@fedora ~]$ ./s3.sh 1
n
[mgandrianova@fedora ~]$ ./s3.sh 1
x
[mgandrianova@fedora ~]$
```

Рис.14: Проверка скрипта

Выводы

Я изучила основы программирования в оболочке ОС UNIX и научилась 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:

seq 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` 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.