

Отчёт по лабораторной работе №12”

Дисциплина: Операционные системы

Батова Ирина Сергеевна, НММбд-01-22

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Выводы	17
5	Контрольные вопросы	18

Список иллюстраций

3.1	Создание файла 'file31.sh'	8
3.2	Программа 1	9
3.3	Проверка корректности программы 1	10
3.4	Измененная программа 1	11
3.5	Измененная программа 1	12
3.6	Проверка корректности измененной программы 1	13
3.7	Каталог со справками о командах	13
3.8	Создание файла 'file32.sh'	13
3.9	Программа 2	14
3.10	Проверка корректности программы 2	14
3.11	Проверка корректности программы 2	15
3.12	Проверка корректности программы 2	15
3.13	Создание файла 'file33.sh'	15
3.14	Программа 3	16
3.15	Проверка корректности программы 3	16

Список таблиц

1 Цель работы

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

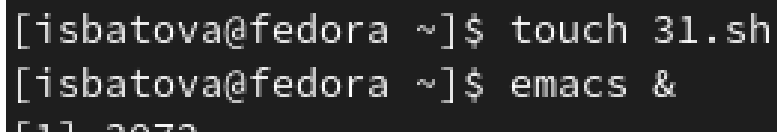
2 Задание

1. Написать 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.
2. Реализовать команду `man` с помощью командного файла. Изучите содержимое каталога `/usr/share/man/man1`. 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`.
3. Используя встроенную переменную `$RANDOM`, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита.

Учтите, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

3 Выполнение лабораторной работы

1. Для начала работы создаем файл для написания скрипта и открываем его (рис. 3.1).



```
[isbatova@fedora ~]$ touch 31.sh  
[isbatova@fedora ~]$ emacs &  
[1] 3073
```

Рис. 3.1: Создание файла 'file31.sh'

Нам необходимо, написать командный файл, реализующий упрощённый механизм семафоров.

Вводим скрипт в наш файл (рис. 3.2).


```
#!/bin/bash

ti=$1
me=$2
s1=$(date +%s)
s2=$(date +%s)
((t=s2-s1))

while ((t<ti))
do
    echo "Ожидание"
    sleep 1
    s2=$(date +%s)
    ((t=s2-s1))
done

s1=$(date +%s)
s2=$(date +%s)
((t=s2-s1))

while ((t<me))
do
    echo "Выполнение"
    sleep 1
    s2=$(date +%s)
    ((t=s2-s1))
done
```

Рис. 3.2: Программа 1

В данном скрипте мы вводим как переменные время ожидания и время выполнения (вводятся пользователем при запуске командного файла), а также два счетчика времени и еще изменяемый счетчик (разница двух предыдущих счетчиков). Далее мы пишем два цикла `while` - для ожидания и для выполнения. Внутри каждого из циклов мы выводим соответствующее сообщение и делаем паузу в 1 секунду для занесения изменений в счетчик. Между циклами мы обновляем все три счетчика для корректной работы второго цикла.

Далее добавляем право на выполнение файла командой `'chmod +x *.sh'` и выпол-

нinem скрипт командой './file31.sh (аргументы)'. Программа работает корректно (рис. 3.3).

```
[isbatova@fedora ~]$ chmod +x *.sh
[isbatova@fedora ~]$ ./file31.sh 7 2
Ожидание
Ожидание
Ожидание
Ожидание
Ожидание
Ожидание
Ожидание
Ожидание
Выполнение
Выполнение
[isbatova@fedora ~]$
```

Рис. 3.3: Проверка корректности программы 1

Далее нам необходимо доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов. Вновь открываем файл и вводим в него измененный скрипт (рис. 3.4, 3.5).

```
#!/bin/bash

function waiting
{
s1=$(date +%s)
s2=$(date +%s)
((t=s2-s1))

while ((t<ti))
do
    echo "Ожидание"
    sleep 1
    s2=$(date +%s)
    ((t=s2-s1))
done
}

function todo
{
s1=$(date +%s)
s2=$(date +%s)
((t=s2-s1))

while ((t<me))
do
    echo "Выполнение"
    sleep 1
    s2=$(date +%s)
    ((t=s2-s1))
done
}
```

Рис. 3.4: Измененная программа 1

```

ti=$1
me=$2
com=$3
while true
do
    if [ $com == Выход ]
    then
        echo "Выход"
        exit 0
    fi
    if [ $com == Ожидание ]
    then waiting
    fi

    if [ $com == Выполнение ]
    then todo
    fi

    echo "Введите следующее действие:"
    read command

done

```

Рис. 3.5: Измененная программа 1

В измененном скрипте мы заносим обновление счетчиков и циклы `while` под функции, и вводим три переменных - время ожидания, время выполнения и переменную-указание к действию. Далее под циклом `while true` рассматриваем три варианта значения переменной-указания к действию с помощью `if` и обращаемся к соответствующей функции (или осуществляем выход). В конце выводим предложение ввести следующей действие и осуществляем аналогичные действия.

Далее выполняем скрипт командой `./file31.sh (аргументы)`. Программа работает корректно (рис. 3.6).

```
[isbatova@fedora ~]$ ./file31.sh 1 2 Ожидание > /dev/tty
Ожидание
Введите следующее действие:
█
```

Рис. 3.6: Проверка корректности измененной программы 1

2. Для начала работы переходим в каталог со справками о командах `‘/usr/share/man/man1’` и с помощью команды `ls` просматриваем, какие файлы в нем содержатся (рис. 3.7).

```
[isbatova@fedora ~]$ cd /usr/share/man/man1
[isbatova@fedora man1]$ ls
.:.1.gz
' [.1.gz'
a2ping.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
```

Рис. 3.7: Каталог со справками о командах

Из данного действия мы получили информацию, что все справки о командах хранятся под именем `*.1.gz`. Это пригодится нам при написании командного файла.

Далее создаем файл для написания скрипта и открываем его (рис. 3.8).

```
[isbatova@fedora ~]$ touch file32.sh
[isbatova@fedora ~]$ emacs &
[1] 4779
```

Рис. 3.8: Создание файла `‘file32.sh’`

Нам необходимо, написать командный файл, реализующий команду man. Вводим скрипт в наш файл (рис. 3.9).

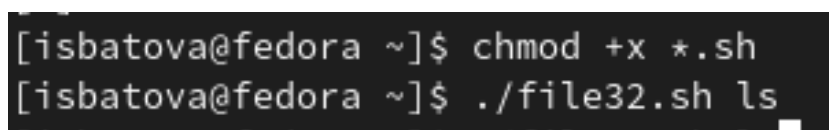
```
#!/bin/bash

a=$1
if [ -f /usr/share/man/man1/$a.1.gz ]
then
    gunzip -c /usr/share/man/man1/$a.1.gz | less
else
    echo "Информация по данной команде не найдена"
fi
```

Рис. 3.9: Программа 2

В данном скрипте мы вводим переменную, которая принимает значение, введенное пользователем при запуске командной файла (название команды). Далее мы проверяем, есть ли информация по данной команде и с помощью if выводим информацию по введенной пользователем команде или сообщение, что информация по данной команде отсутствует.

Далее добавляем право на выполнение файла командой 'chmod +x *.sh' и выполняем скрипт командой './file32.sh (аргумент)'. Программа работает корректно как при введении названия существующей команды (рис. 3.10, 3.11), так и не существующей (рис. 3.12).



```
[isbatova@fedora ~]$ chmod +x *.sh
[isbatova@fedora ~]$ ./file32.sh ls
```

Рис. 3.10: Проверка корректности программы 2

```

.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.48.5.
.TH LS "1" "August 2022" "GNU coreutils 9.1" "User Commands"
.SH NAME
ls \- list directory contents
.SH SYNOPSIS
.B ls
[\fI\,OPTION\|\fR]... [\fI\,FILE\|\fR]...
.SH DESCRIPTION
.\" Add any additional description here
.PP
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of \fB\--cftuvSUX\fR nor \fB\--sort\fR is sp
ecified.
.PP
Mandatory arguments to long options are mandatory for short options too.
.TP
\fB\--a\fR, \fB\--all\fR
do not ignore entries starting with .
.TP
\fB\--A\fR, \fB\--almost-all\fR
do not list implied . and ..
.TP
\fB\--author\fR

```

Рис. 3.11: Проверка корректности программы 2

```

[isbatova@fedora ~]$ chmod +x *.sh
[isbatova@fedora ~]$ ./file32.sh ls
[isbatova@fedora ~]$ ./file32.sh mpmr
Информация по данной команде не найдена

```

Рис. 3.12: Проверка корректности программы 2

3. Для начала работы создаем файл для написания скрипта и открываем его (рис. 3.13).

```

[isbatova@fedora ~]$ touch file33.sh
[isbatova@fedora ~]$ emacs &
[1] 4990

```

Рис. 3.13: Создание файла 'file33.sh'

Нам необходимо, используя встроенную переменную \$RANDOM, написать командный файл, генерирующий случайную последовательность букв латинского алфавита.

Вводим скрипт в наш файл (рис. 3.14).

```
#!/bin/bash
a=$1
for (( i=0; i<$a; i++))
do
    ((b=$RANDOM%26+1))
    case $b 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
```

Рис. 3.14: Программа 3

В данном скрипте мы вводим переменную, которая принимает значение, введенное пользователем при запуске командной файла (количество символов). Далее с помощью цикла for мы выводим нужное количество символов. Внутри цикла используется встроенная переменная \$RANDOM для определения случайного номера и команда case для непосредственного вывода символа (каждая команда вывода символа обозначается под своим порядковым номером, который и выбирает встроенная переменная \$RANDOM).

Далее добавляем право на выполнение файла командой 'chmod +x *.sh' и выполняем скрипт командой './file33.sh (аргументы)'. Для проверки корректности выполнения вводим несколько чисел (рис. 3.15).

```
[isbatova@fedora ~]$ chmod +x *.sh
[isbatova@fedora ~]$ ./file33.sh 6
ulphej
[isbatova@fedora ~]$ ./file33.sh 23
lumazndwutloqaahoxrrgqq
```

Рис. 3.15: Проверка корректности программы 3

4 Выводы

В ходе данной лабораторной работы мной были изучены основы программирования в оболочке ОС UNIX. Помимо этого, я научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 Контрольные вопросы

1. В данной строке не хватает пробелов после первой скобки и перед второй, а также кавычек у первой переменной.

Правильный вариант:

```
while [ "$1" != "exit" ]
```

2. Есть два способа объединения несколько строк в одну.
3. `a="hi" b="bye" c="ab" echo "$c"`
4. `a="hi" a+="bye" echo "$a"`
5. Утилита `seq` используется для генерации чисел от первого до последнего шага `INCREMENT`.
6. Результат даст вычисление выражения `$((10/3))` равен 3.
7. Отличия `zsh`:
 - более быстрое автодополнение для `cd` с помощью `Tab`
 - есть калькулятор, способный выполнять вычисления внутри терминала
 - поддерживается раскрытие полного пути на основе неполных данных
 - поддерживается замена части пути
6. У конструкции `for ((a=1; a <= LIMIT; a++))` верен синтаксис.
7. Преимущества `bash`:

- один из самых популярных языков программирования
- удобное перенаправление ввода/вывода
- большое количество команд для работы с файловыми системами

Недостатки bash:

- небольшая библиотека относительно других языков
- достаточно медленный, так как утилиты, при выполнении скрипта, запускают свои процессы
- скрипты нельзя запустить на других операционных системах без дополнительных действий