

Отчёт по лабораторной работе №13

Дисциплина: Операционные системы

Аветисян Давид Артурович

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Контрольные вопросы	18
5	Выводы	21

Список таблиц

Список иллюстраций

3.1	Создал файл и написал первый скрипт	9
3.2	Проверил первый скрипт	10
3.3	Изменил первый скрипт	11
3.4	Изменил первый скрипт	12
3.5	Проверил первый скрипт повторно	13
3.6	Содержимое каталога /usr/share/man/man1	14
3.7	Создал файл и написал второй скрипт	14
3.8	Проверил второй скрипт	15
3.9	Проверил второй скрипт	15
3.10	Проверил второй скрипт	16
3.11	Создал файл и написал третий скрипт	16
3.12	Проверил третий скрипт	17

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. Написал 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 данной задачи я создал файл: `semafor.sh` и написал соответствующий скрипт (рис. -fig. 3.1).


```
#!/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
```

Рис. 3.1: Создал файл и написал первый скрипт

Далее я проверил работу написанного скрипта (команда «./semafor.sh 3 5»), предварительно добавив право на исполнение файла (команда «chmod +x semafor.sh») (рис. -fig. 3.2). Скрипт работает корректно.

```
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ chmod +x semafor.sh
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ ./semafor.sh
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ ./semafor.sh 3 5
Ожидание
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
```

Рис. 3.2: Проверил первый скрипт

После этого я изменил скрипт так, чтобы его можно было выполнять в нескольких терминалах (рис. -fig. 3.3) (рис. -fig. 3.4) и проверил его работу (например, команда «./semafor.sh 2 4 Ожидание > /dev/pts/1 &»).

```
#!/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
}
```

Рис. 3.3: Изменил первый скрипт

```
t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then ozhidanie
    fi
    if [ "$command" == "Выполнение" ]
    then vipolnenie
    fi
    echo "Следующее действие: "
    read command
done
```

Рис. 3.4: Изменил первый скрипт

Но ни одна команда не работала, так как мне было “Отказано в доступе” (рис. -fig. 3.5). При этом скрипт работает корректно (команда «./semafor.sh 2 4 Ожидание»).

```

daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ ./semafor.sh 2 4 Ожидание > &
bash: синтаксическая ошибка рядом с неожиданным маркером «&»
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ sudo ./semafor.sh 2 4 Ожидание > /dev/pts/1 &
[2] 6102
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ bash: /dev/pts/1: Отказано в доступе

[2]+ Выход 1          sudo ./semafor.sh 2 4 Ожидание > /dev/pts/1
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ ./semafor.sg 2 4 Ожидание > /dev/pts/1 &
[2] 6104
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ bash: /dev/pts/1: Отказано в доступе

[2]+ Выход 1          ./semafor.sg 2 4 Ожидание > /dev/pts/1
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ ./semafor.sg 3 4 Ожидание > /dev/pts/2 &
[2] 6106
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ bash: /dev/pts/2: Отказано в доступе

[2]+ Выход 1          ./semafor.sg 3 4 Ожидание > /dev/pts/2
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ ./semafor.sg 2 5 Выполнение > /dev/pts/2 &
[2] 6109
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ bash: /dev/pts/2: Отказано в доступе

[2]+ Выход 1          ./semafor.sg 2 5 Выполнение > /dev/pts/2
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ ./semafor.sg 3 5 Выход > /dev/pts/2 &
[2] 6110
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ bash: /dev/pts/2: Отказано в доступе

[2]+ Выход 1          ./semafor.sg 3 5 Выход > /dev/pts/2
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ ./semafor.sg 2 6 Выполнение > /dev/pts/3 &
[2] 6112
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ bash: /dev/pts/3: Отказано в доступе

[2]+ Выход 1          ./semafor.sg 2 6 Выполнение > /dev/pts/3

```

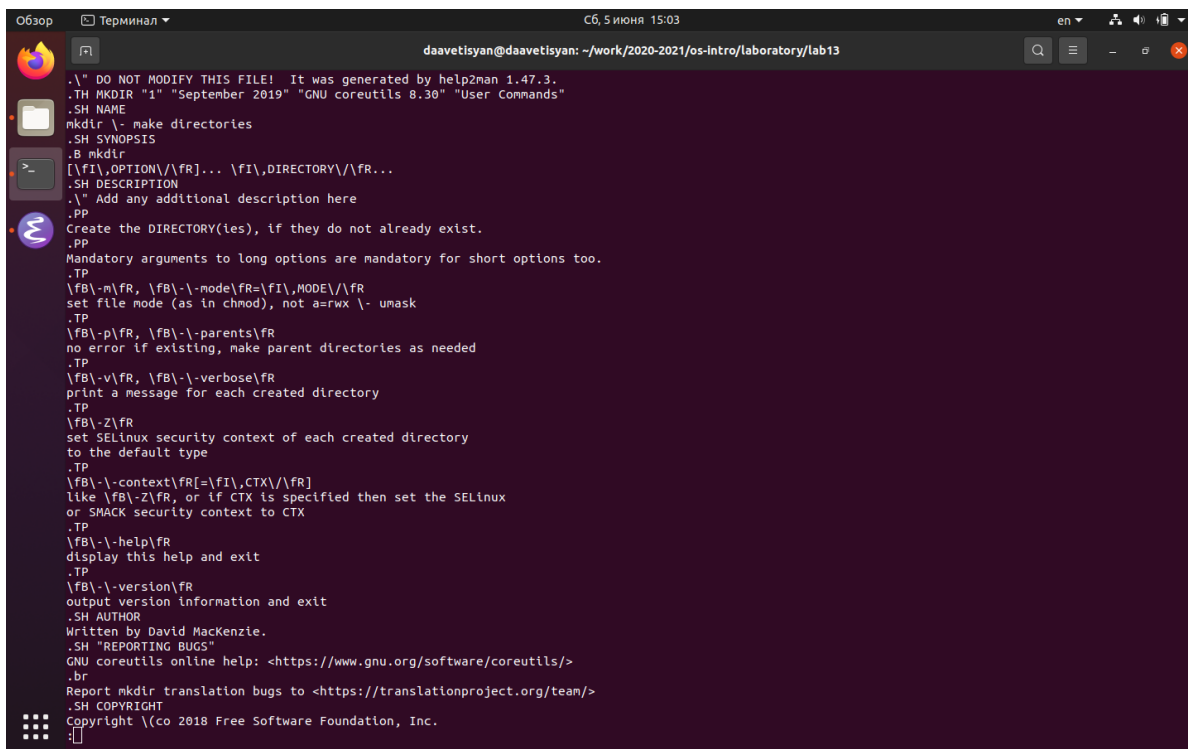
Рис. 3.5: Проверил первый скрипт повторно

2. Реализовал команду `man` с помощью командного файла. Изучил содержимое каталога `/usr/share/man/man1` (рис. -fig. 3.6). 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`.

Скрипт работает корректно.

```
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ chmod +x man.sh
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ ./man.sh mkdir
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ ./man.sh rm
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ ./man.sh car
Справки по данной команде нет
```

Рис. 3.8: Проверил второй скрипт



```
Обзор Терминал C6, 5 июня 15:03 en
daavetisyan@daavetisyan: ~/work/2020-2021/os-intro/laboratory/lab13

.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH MKDIR "1" "September 2019" "GNU coreutils 8.30" "User Commands"
.SH NAME
mkdir \- make directories
.SH SYNOPSIS
.B mkdir
[FI],OPTION[\i>FR]... [FI],DIRECTORY[\i>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
\i>FB \-m\i>FR, \i>FB \-mode\i>FR=FI \i>MODE[\i>FR]
set file mode (as in chmod), not a=rwx \- umask
.TP
\i>FB \-p\i>FR, \i>FB \-parents\i>FR
no error if existing, make parent directories as needed
.TP
\i>FB \-v\i>FR, \i>FB \-verbose\i>FR
print a message for each created directory
.TP
\i>FB \-Z\i>FR
set SELinux security context of each created directory
to the default type
.TP
\i>FB \--context\i>FR[=FI \i>CTX][\i>FR]
like \i>FB \-Z\i>FR, or if CTX is specified then set the SELinux
or SMACK security context to CTX
.TP
\i>FB \-help\i>FR
display this help and exit
.TP
\i>FB \-version\i>FR
output version information and exit
.SH AUTHOR
Written by David MacKenzie.
.SH "REPORTING BUGS"
GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
.br
Report mkdir translation bugs to <https://translationproject.org/team/>
.SH COPYRIGHT
Copyright \(\co 2018 Free Software Foundation, Inc.
:[]
```

Рис. 3.9: Проверил второй скрипт

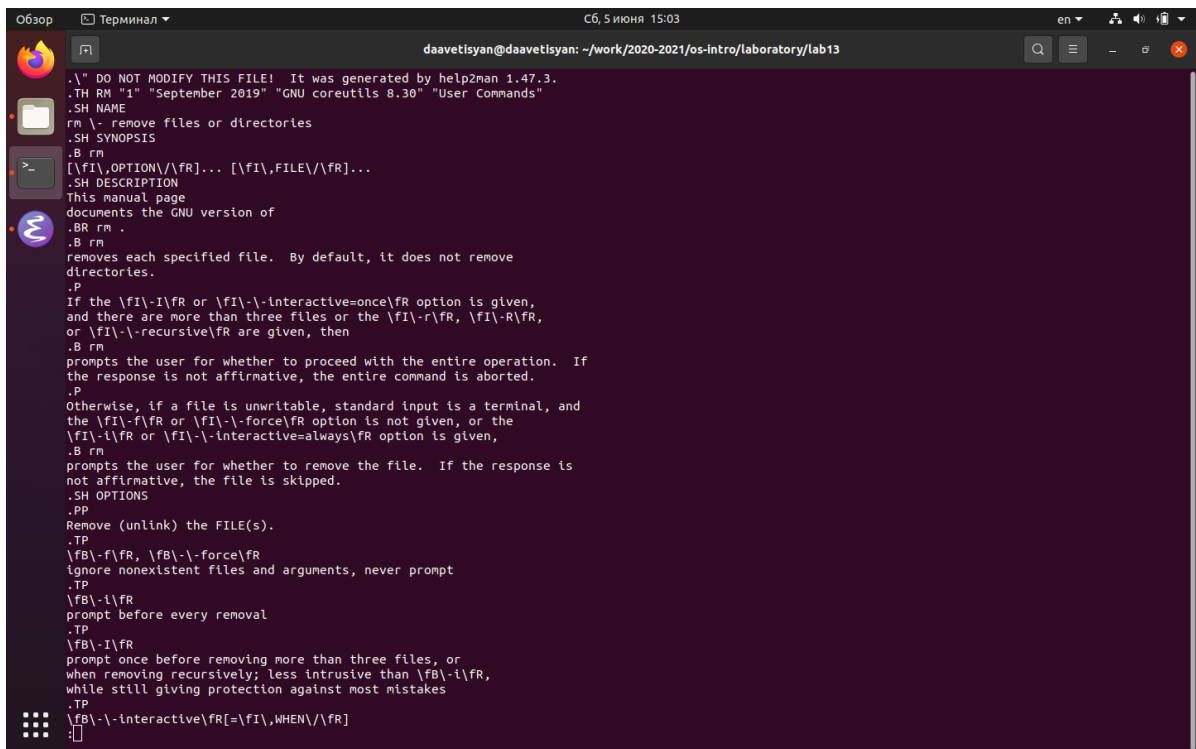


Рис. 3.10: Проверил второй скрипт

- Используя встроенную переменную \$RANDOM, написал командный файл, генерирующий случайную последовательность букв латинского алфавита. Для данной задачи я создал файл: random.sh и написал соответствующий скрипт (рис. -fig. 3.11).

```
#!/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
```

Рис. 3.11: Создал файл и написал третий скрипт

Далее я проверил работу написанного скрипта (команда «./random.sh 15»), предварительно добавив право на исполнение файла (команда «chmod +x random.sh») (рис. -fig. 3.12). Скрипт работает корректно.


```
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ chmod +x random.sh
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab13$ ./random.sh 15
ynyamotygnozvkh
```

Рис. 3.12: Проверил третий скрипт

4 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="VAR1VAR2"
```

```
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` 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

5 Выводы

В ходе выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX, а также научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.