

Отчет по лабораторной работе 12

Операционные системы

Скрипникова София

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Контрольные вопросы	16
5	Выводы	19

Список иллюстраций

3.1	Создание файла	8
3.2	Создание файла	9
3.3	Результат работы программы	10
3.4	Текст	11
3.5	Текст	12
3.6	Результат работы программы	13
3.7	Создание файла	13
3.8	Создание файла	13
3.9	Результат работы программы	14
3.10	Создание файла	14
3.11	Создание файла	15
3.12	Результат работы программы	15

Список таблиц

1 Цель работы

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

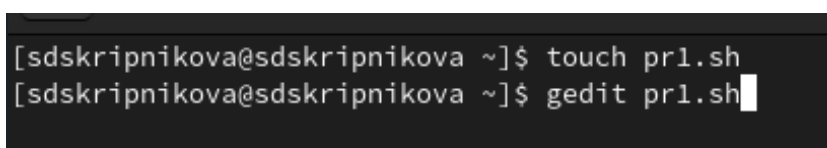
2 Задание

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. Создала файл для программы 1 (рис. 3.1).



```
[sdskripnikova@sdskripnikova ~]$ touch pr1.sh  
[sdskripnikova@sdskripnikova ~]$ gedit pr1.sh
```

Рис. 3.1: Создание файла

2. Написала текст программы 1 (рис. 3.2).


```
1 #!/bin/bash
2 t1=$1
3 t2=$2
4 s1=$(date +%s)
5 s2=$(date +%s)
6 ((t=s2 - s1))
7 while ((t<t1))
8 do
9     echo "Ожидание"
10    sleep 1
11    s2=$(date +%s)
12    ((t=s2 -s1))
13 done
14 s1=$(date +%s)
15 s2=$(date +%s)
16 ((t=s2 - s1))
17 while ((t<t2))
18 do
19     echo "Выполнение"
20    sleep 1
21    s2=$(date +%s)
22    ((t=s2 -s1))
23 done
```

Рис. 3.2: Создание файла

3. Проверила работу написанной программы (рис. 3.3).

```
[sdscripnikova@sdscripnikova ~]$ chmod +x pr1.sh
[sdscripnikova@sdscripnikova ~]$ ./pr1.sh 3 5
Ожидание
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
[sdscripnikova@sdscripnikova ~]$
```

Рис. 3.3: Результат работы программы

4. Изменила текст прогаммы 1 (рис. 3.4), (рис. 3.5).

```
1 #!/bin/bash
2
3 function ozhidanie
4
5 {
6
7     s1=$(date +%s)
8
9     s2=$(date +%s)
10
11     ((t=s2 - s1))
12
13     while ((t<t1))
14
15     do
16
17     echo "Ожидание"
18
19     sleep 1
20
21     s2=$(date +%s)
22
23     ((t=s2 -s1))
24
25     done
26
27 }
28
29 function vipolnenie
30
31 {
32
33     s1=$(date +%s)
34
35     s2=$(date +%s)
36
37     ((t=s2 - s1))
38
```

Рис. 3.4: Текст

```
54
55 t1=$1
56
57 t2=$2
58
59 command=$3
60
61 while true
62 do
63
64     if [ "$command" == "Выход" ]
65     then
66
67         echo "Выход"
68         exit 0
69     fi
70
71     if [ "$command" == "Ожидание" ]
72     then ozhidanie
73     fi
74
75     if [ "$command" == "Выполнение" ]
76     then vipolnenie
77     fi
78
79     echo "Следующее действие: "
80
81     read command
82
83 done
```

Рис. 3.5: Текст

5. Проверила работу написанной программы (рис. 3.6).

```
[sdscripnikova@sdscripnikova ~]$ gedit pr1.sh
[sdscripnikova@sdscripnikova ~]$ ./pr1.sh 3 5
Следующее действие:
ожидание
Следующее действие:
выполнение
Следующее действие:
█
```

Рис. 3.6: Результат работы программы

6. Создала файл для программы 2 (рис. 3.7).

```
[sdscripnikova@sdscripnikova ~]$ touch pr2.sh
[sdscripnikova@sdscripnikova ~]$ gedit pr2.sh █
```

Рис. 3.7: Создание файла

8. Написала текст программы 2 (рис. 3.8).

```
1 #!/bin/bash
2
3 a=$1
4
5 if [ -f /usr/share/man/man1/$a.1.gz ]
6
7 then
8
9     gunzip -c /usr/share/man/man1/$1.1.gz | less
10
11 else
12
13     echo "Справки по данной команде нет"
14
15 fi
```

Рис. 3.8: Создание файла

9. Проверила работу написанной программы (рис. 3.9).

```

.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.48.5.
.TH MKDIR "1" "January 2023" "GNU coreutils 9.0" "User Commands"
.SH NAME
mkdir \- make directories
.SH SYNOPSIS
.B mkdir
[\fI\,OPTION\|\fR]... \fI\,DIRECTORY\|\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\m\mode\fR=\fI\,MODE\|\fR
set file mode (as in chmod), not a=rwx \- umask
.TP
\fB\p\fR, \fB\p\parents\fR
no error if existing, make parent directories as needed,
with their file modes unaffected by any \fB\m\fR option.
.TP
\fB\v\fR, \fB\v\verbose\fR
be verbose, outputting a message for each directory created.

```

Рис. 3.9: Результат работы программы

10. Создала файл для программы 3 (рис. 3.10).

```

[sdskripnikova@sdskripnikova ~]$ touch pr3.sh
[sdskripnikova@sdskripnikova ~]$ gedit pr3.sh

```

Рис. 3.10: Создание файла

11. Написала текст программы 3 (рис. 3.11).

```

1 #!/bin/bash
2
3 a=$1
4
5 for ((i=0; i<$a; i++))
6
7 do
8
9     ((char=$RANDOM%26+1))
10
11     case $char in
12
13 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)
14 echo -n h;; 9) echo -n i;;
15 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
16 p;; 17) echo -n q;;
17 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
18 x;; 25) echo -n y;;
19 26) echo -n z;;
20
21     esac
22
23 done
24
25 echo

```

Рис. 3.11: Создание файла

12. Проверила работу написанной программы (рис. 3.12).

```

[sdskripnikova@sdskripnikova ~]$ chmod +x pr3.sh
[sdskripnikova@sdskripnikova ~]$ ./pr3.sh 15
nfflvroypmvgsrv
[sdskripnikova@sdskripnikova ~]$

```

Рис. 3.12: Результат работы программы

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 изучила основы программирования в оболочке ОС UNIX, а также научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.