

Лабораторная работа №13

Операционные системы

Тойчубекова Асель Нурлановна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Ответы на вопросы	15
6	Выводы	18

Список иллюстраций

4.1	Создание файла	9
4.2	Редактирование файла	10
4.3	Запуск программы	10
4.4	Создание файла	10
4.5	Справки о командах	11
4.6	Редактирование файла	12
4.7	Запуск программы	12
4.8	Создание файла	12
4.9	Редактирование файла	13
4.10	Запуск программы	14

Список таблиц

1 Цель работы

Целью данной лабораторной рабораторной работы является изучение основ программирования в оболочке ОС Unix. Также 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, 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 Теоретическое введение

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:

- оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
- C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

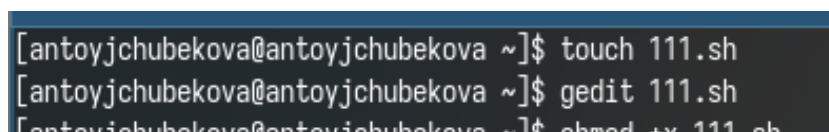
Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-

подобных операционных систем и переносимости прикладных программ на уровне исходного кода.

POSIX-совместимые оболочки разработаны на базе оболочки Корна.

4 Выполнение лабораторной работы

Для начала создаю файл в котором буду писать саму программу и открываю его в редакторе gedit (рис. 4.1).



```
[antoyjchubekova@antoyjchubekova ~]$ touch 111.sh
[antoyjchubekova@antoyjchubekova ~]$ gedit 111.sh
[antoyjchubekova@antoyjchubekova ~]$ chmod +x 111.sh
```

Рис. 4.1: Создание файла

Редактирую файл, записывая командный файл, который реализует упрощенный механизм семафоров. Для этого я сперва создаю переменную в которой будет храниться адрес файла блокирования, а также переменную, которому присваивается значение первого аргумента. Далее открыв файл блокирования с помощью команды exes и приваиваю ему свободный файловый дескриптор. Далее запускаю цикл, который будет работать пока файл блокирования существует и проверяю можно ли заблокировать файл с помощью команды flock -n, если да, то вывожу сообщение, что файл заблокирован, затем жду столько времени, сколько было указано в аргументе и разблокирую файл с помощью команды flock -u, выведя об этом сообщение. Если же файл изначально был заблокирован, то я вывожу сообщения, что файл заблокирован и жду пока не истечет время, указанное в аргументе. (рис. 4.2).

```

1 #!/bin/bash
2 lock_file="/.lock.file"
3 a=1
4 exec {fn}>$lock_file
5 while test -f "$lock_file"
6 do
7 if flock -n ${fn}
8 then
9 echo Файл заблокирован \(\Ресурс используется другим процессом\)\!
10 sleep $a
11 echo Файл разблокирован \(\Ресурс готов к использованию\)\!
12 flock -u ${fn}
13 else
14 echo Файл заблокирован \(\Ресурс используется другим процессом\)\!
15 sleep $a
16 fi
17 done

```

Рис. 4.2: Редактирование файла

Даю права на выполнение и запускаю программу, мы видим, что все работает корректно. (рис. 4.3).

```

[antoyjchubekova@antoyjchubekova ~]$ chmod +x 111.sh
[antoyjchubekova@antoyjchubekova ~]$ bash 111.sh 5
Файл заблокирован (Ресурс используется другим процессом)!
Файл разблокирован (Ресурс готов к использованию)!
Файл заблокирован (Ресурс используется другим процессом)!
Файл разблокирован (Ресурс готов к использованию)!
Файл заблокирован (Ресурс используется другим процессом)!
Файл разблокирован (Ресурс готов к использованию)!

```

Рис. 4.3: Запуск программы

Создаю файл для написания программы второго задания. (рис. 4.4).

```

[antoyjchubekova@antoyjchubekova ~]$ gedit 111.sh
[antoyjchubekova@antoyjchubekova ~]$ touch 222.sh

```

Рис. 4.4: Создание файла

Далее перехожу по ссылке `/usr/share/man/man1` и вижу архивы текстовых файлов, содержащих справки о командах. (рис. 4.5).

```

ffprobe.1.gz
ffprobe-all.1.gz
fg.1.gz
fgconsole.1.gz
file.1.gz
file2brl.1.gz
filterdiff.1.gz
fincore.1.gz
find.1.gz
findhyph.1.gz
findrule.1.gz
firefox.1.gz
firewall-cmd.1.gz
firewalld.1.gz
firewall-offline-cmd.1.gz
fish.1.gz
fish_indent.1.gz
fish_key_reader.1.gz
fixcvsdiff.1.gz
flex++.1.gz
flex.1.gz
flexiblas.1.gz
flipdiff.1.gz
flock.1.gz
fmt.1.gz
fmtutil.1.gz
fmtutil-sys.1.gz
fmtutil-user.1.gz
fold.1.gz
fontforge.1.gz
fontimage.1.gz
fontinst.1.gz
fontlint.1.gz
foomatic-rip.1.gz
foot.1.gz
footclient.1.gz
hunspell.1.gz
hyperxmp-add-bytecount.1.gz
iasecc-tool.1.gz
iconv.1.gz
id.1.gz
identify.1.gz
ieccset.1.gz
ImageMagick.1.gz
implantisomd5.1.gz
import.1.gz
imv.1.gz
imv-dir.1.gz
imv-msg.1.gz
imv-wayland.1.gz
imv-x11.1.gz
includes.1.gz
info.1.gz
info.1lossl.gz
infocmp.1m.gz
infotocap.1m.gz
inimf.1.gz
init.1.gz
initex.1.gz
inkscape.1.gz
install.1.gz
install-info.1.gz
install-tl.1.gz
interdiff.1.gz
intro.1.gz
ionice.1.gz
ipcalc.1.gz
ipcmk.1.gz
ipcrm.1.gz
ipcs.1.gz
ippfind.1.gz
ipptool.1.gz

```

Рис. 4.5: Справки о командах

Открываю созданный файл в редакторе и редактирую его, записывая командный файл, который реализует команду `man`, перейдя по адресу `/usr/share/man/man1` и используя архивы данных о командах выполняет эту команду. Для этого сперва завожу переменную, которой присваиваю значения аргумента, затем проверяю есть ли подходящий файл в `man1`, если есть с помощью команды `less` вывожу ее, а если нет вывожу соответствующее сообщение. (рис. 4.6).

```

1 #!/bin/bash
2 a=$1
3 if test -f /usr/share/man/man1/$a.1.gz
4 then
5 less /usr/share/man/man1/$a.1.gz
6 else
7 echo Описание команды не найдено\!
8 fi

```

Рис. 4.6: Редактирование файла

Даю права на выполнение и запускаю программу, мы видим, что все работает правильно и программа выдает нам сообщение о команде. (рис. 4.8).

```

ESC[4mCPESC[24m(1)
ESC[4mCPESC[24m(1)

ESC[1mNAMEESC[0m
cp - copy files and directories

ESC[1mSYNOPSISESC[0m
ESC[1mcp ESC[22mESC[4mOPTIONESC[24m]... [ESC[4m-]ESC[24m] ESC[4mSOURCEESC[24m ESC[4mDESTESC[0m
ESC[1mcp ESC[22mESC[4mOPTIONESC[24m]... ESC[4mSOURCEESC[24m... ESC[4mDIRECTORYESC[0m
ESC[1mcp ESC[22mESC[4mOPTIONESC[24m]... ESC[4m-ESC[24m ESC[4mDIRECTORYESC[24m ESC[4mSOURCEESC[24m...

ESC[1mDESCRIPTIONESC[0m
Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

Mandatory arguments to long options are mandatory for short options too.

ESC[1m-ESC[22m, ESC[1m--archiveESC[0m
same as ESC[1m-dR --preserveESC[22mESC[4mallESC[0m

ESC[1m--attributes-onlyESC[0m
don't copy the file data, just the attributes

ESC[1m--backupESC[22m[=ESC[4mCONTROLESC[24m]
make a backup of each existing destination file

ESC[1m-b ESC[22mlike ESC[1m--backup ESC[22mbut does not accept an argument

ESC[1m--copy-contentsiESC[0m
copy contents of special files when recursive

```

Рис. 4.7: Запуск программы

Создаю файл для написания программы для третьего задания и открываю его в редакторе. (рис. 4.8).

```

[antoyjchubekova@antoyjchubekova ~]$ touch 333.sh
[antoyjchubekova@antoyjchubekova ~]$ gedit 333.sh

```

Рис. 4.8: Создание файла

Редактирую файл, записывая командный файл, который используя встроенную переменную \$RANDOM, генерирует случайную последовательность букв латинского алфавита. Для этого я создаю переменную и присваиваю ей значения аргумента, далее запускаю цикл, который будет продолжаться пока i не станет равным аргументу, запускаю RANDOM и в соответствии с получившимся числом с помощью команды case перебираю по цифрам все буквы, если совпадает вывожу на экран, в конце перехожу на новую строку. (рис. 4.9).

```
1 #!/bin/bash
2 a=$1
3 for((i=0;i<=$a;i++))
4 do
5 rnum=$((RANDOM % 26+1))
6 case $rnum in
7 1) echo -n a;;
8 2) echo -n b;;
9 3) echo -n c;;
10 4) echo -n d;;
11 5) echo -n e;;
12 6) echo -n f;;
13 7) echo -n g;;
14 8) echo -n h;;
15 9) echo -n i;;
16 10) echo -n j;;
17 11) echo -n k;;
18 12) echo -n l;;
19 13) echo -n m;;
20 14) echo -n n;;
21 15) echo -n o;;
22 16) echo -n p;;
23 17) echo -n q;;
24 18) echo -n r;;
25 19) echo -n s;;
26 20) echo -n t;;
27 21) echo -n u;;
28 22) echo -n v;;
29 23) echo -n 'w';;
30 24) echo -n x;;
31 25) echo -n y;;
32 26) echo -n z;;
33 esac
34 done
35 echo
36
```

Рис. 4.9: Редактирование файла

Даю права на выполнение и запускаю программу, мы видим, что программа правильно работает и выводит латинские буквы. (рис. 4.10).

```
antoyjchubekova@antoyjchubekova ~]$ chmod +x 333.sh
antoyjchubekova@antoyjchubekova ~]$ bash 333.sh 5
piueh
antoyjchubekova@antoyjchubekova ~]$ bash 333.sh 10
cbfptvxcrb
antoyjchubekova@antoyjchubekova ~]$ bash 333.sh 86
tiasfjenugoxbedhujdafpfwvocwcyibaapineycuxenwuqeeyyr1jlpplinyssftd1gadfvntxexkhqmaugr
antoyjchubekova@antoyjchubekova ~]$
```

Рис. 4.10: Запуск программы

5 Ответы на вопросы

1. 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.
2. Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами: Первый: VAR1=“Hello,” VAR2=“ World” VAR3=“VAR1VAR2” echo “VAR3” : *Hello,World* : 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 и недостатки 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;

6 Выводы

В ходе лабораторной работы №14 я изучила основы программирования в оболочке ОС Unix. Также научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.