

Отчёт по лабораторной работе №14

Операционные системы

Луангсуваннавонг Сайпхачан

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	8
4	Выполнение лабораторной работы	10
5	Выводы	18
6	Ответы на контрольные вопросы	19
	Список литературы	21

Список иллюстраций

4.1	Создание файла	10
4.2	Код программы	11
4.3	Запуск программы	13
4.4	Директория /usr/share/man/man1	14
4.5	Создание файла и код программы	15
4.6	Запуск программы	15
4.7	информация о команде	16
4.8	Создание файла и код программы	16
4.9	Запуск программы	17

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 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:

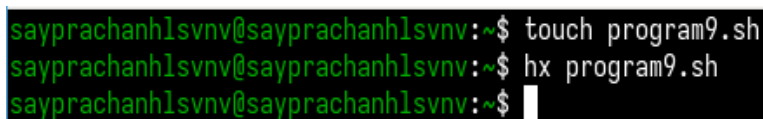
- оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
- С-оболочка (или csh) — надстройка на оболочкой Борна, использующая С-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С 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. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с опи-

санными ниже.

4 Выполнение лабораторной работы

Сначала я создаю файл 'program9.sh' и открываю его в текстовом редакторе для редактирования. (рис. 4.1)

A terminal window with a black background and green text. It shows three lines of commands being executed in a shell. The first line creates a file named 'program9.sh' using the 'touch' command. The second line opens the file in a text editor named 'hx'. The third line shows the prompt after the editor has been opened.

```
sayprachanh1svnv@sayprachanh1svnv:~$ touch program9.sh
sayprachanh1svnv@sayprachanh1svnv:~$ hx program9.sh
sayprachanh1svnv@sayprachanh1svnv:~$
```

Рис. 4.1: Создание файла

В файле 'program9.sh' я ввожу программу, которая реализует упрощённый механизм семафора с помощью команды 'flock'. Она ожидает некоторое время для ресурса (я установил это время на 5 секунд (t1)), выводя сообщение об этом, а также время использования ресурса (я установил его на 7 секунд (t2)), и даёт информацию о том, что ресурс используется соответствующим пакетом (процессом). (рис. 4.2)

```

1  #!/bin/bash
2
3  t1=5
4  t2=7
5  lockfile="./lockfile"
6
7  while true;
8  do
9  exec {fn}>$lockfile
10
11  if flock -n $fn
12  then
13      echo "Resource using..."
14      sleep $t2
15      echo "Complete! release resource"
16      flock -u $fn
17
18      break
19  else
20      echo "Waiting for resource..."
21      sleep $t1
22  fi
23  done

```

Рис. 4.2: Код программы

Код программы:

```
#!/bin/bash
```

```
t1=5
```

```
t2=7
```

```
lockfile="./lockfile"
```

```
while true;
```

```
do
exec {fn}>$lockfile

if flock -n $fn
then
    echo "Resource using..."
    sleep $t2
    echo "Complete! release resource"
    flock -u $fn

    break
else
    echo "Waiting for resource..."
    sleep $t1
fi
done
```

После этого я сохраняю файл, даю файлу права на выполнение и запускаю программу. Открываю другой терминал и запускаю ту же команду. Программа во втором терминале ожидает завершения работы программы в первом терминале, затем выводит сообщение, и только после завершения первой программы начинает использовать ресурс и завершает процесс, то есть программа работает корректно. (рис. 4.3)

```
sayprachanhlsnv@sayprachanhlsnv:~$ chmod +x program9.sh
sayprachanhlsnv@sayprachanhlsnv:~$ ./program9.sh
Resource using...
Complete! release resource
sayprachanhlsnv@sayprachanhlsnv:~$ █

Terminal - sayprachanhlsnv@sayprachanhlsnv: /home/sayprachanhlsnv
File Edit View Terminal Tabs Help
sayprachanhlsnv@sayprachanhlsnv:~$ ./program9.sh
Waiting for resource...
Waiting for resource...
Resource using...
Complete! release resource
sayprachanhlsnv@sayprachanhlsnv:~$ █
```

Рис. 4.3: Запуск программы

Далее я проверяю содержимое директории `/usr/share/man/man1` для реализации следующей программы. (рис. 4.4)

```

mkmod.1.gz
mkocp.1.gz
mkofm.1.gz
mkpasswd.1.gz
mkrfc2734.1.gz
mksquashfs.1.gz
mktemp.1.gz
mktexfmt.1.gz
mktexlsr.1.gz
mktexmf.1.gz
mktexpk.1.gz
mktextfm.1.gz
mlabel.1.gz
mmafm.1.gz
mmcli.1.gz
mmd.1.gz
mmdblookup.1.gz
mmount.1.gz
mmove.1.gz
mmpfb.1.gz
modulemd-validator.1.gz
moggsplit.1.gz
mogrify.1.gz
mokutil.1.gz
montage.1.gz
more.1.gz
mount.ddi.1.gz
mountpoint.1.gz
mpage.1.gz
mpartition.1.gz
mpost.1.gz
mptopdf.1.gz
mpv.1.gz
mrd.1.gz
mren.1.gz
xzcat.1.gz
xzcmp.1.gz
xzdec.1.gz
xzdifff.1.gz
xzegrep.1.gz
xzfgrep.1.gz
xzgrep.1.gz
xzless.1.gz
xzmore.1.gz
yes.1.gz
ypdomainname.1.gz
yt-dlp.1.gz
yum-changelog.1.gz
zcat.1.gz
zcmp.1.gz
zdifff.1.gz
zenity.1.gz
zforce.1.gz
zgrep.1.gz
zip.1.gz
zipcloak.1.gz
zipdetails.1.gz
zipgrep.1.gz
zipinfo.1.gz
zipnote.1.gz
zipsplit.1.gz
zless.1.gz
zmore.1.gz
znew.1.gz
zsoelim.1.gz
zvbi-atsc-cc.1.gz
zvbi-chains.1.gz
zvid.1.gz
zvbi-ntsc-cc.1.gz
sayprachanhlsnv@sayprachanhlsnv:~$

```

Рис. 4.4: Директория /usr/share/man/man1

Я создаю файл 'program10.sh' и ввожу программу, которая работает как команда man, она принимает имя команды в качестве аргумента командной строки, а затем с помощью команд 'less' и 'zcat' выводит справочную информацию о команде, указанной в аргументе командной строки. Программа также выведет сообщение, если такой команды нет в директории man/man1. (рис. 4.5)

```

1  #!/bin/bash
2
3  a=$1
4  if test -f /usr/share/man/man1/$a.1.gz
5  then zcat /usr/share/man/man1/$a.1.gz | less
6  else
7    echo "There is no such command"
8  fi
~

```

Рис. 4.5: Создание файла и код программы

Код программы:

```

#!/bin/bash

a=$1
if test -f /usr/share/man/man1/$a.1.gz
then zcat /usr/share/man/man1/$a.1.gz | less
else
echo "There is no such command"
fi

```

Я сохраняю файл, даю файлу права на выполнение и запускаю программу, вводя команду в качестве аргумента. Программа выводит справочную информацию по этой команде. (рис. 4.6 и рис. 4.7)

```

sayprachanhlsnv@sayprachanhlsnv:~$ chmod +x program10.sh
sayprachanhlsnv@sayprachanhlsnv:~$ ./program10.sh mkdir

```

Рис. 4.6: Запуск программы

```

.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.48.5.
.TH MKDIR "1" "November 2024" "GNU coreutils 9.5" "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\-\mode\|\fR=\fI\,MODE\|\fR
set file mode (as in chmod), not a=rwx \- umask

```

Рис. 4.7: информация о команде

Затем я создаю файл 'program11.sh' и ввожу программу, которая генерирует случайную последовательность букв латинского алфавита, используя встроенную переменную \$RANDOM. Количество генерируемых букв будет задано в качестве аргумента командной строки. (рис. 4.8)

```

1 a=$1
2 letter=(a b c d e f g h i j k l m n o p q r s t u v w x y z)
3
4 for ((i=0; i<a; i++))
5 do
6     random=$((RANDOM % 26))
7
8     echo -n "${letter[$random]}"
9 done
10 echo
~

```

Рис. 4.8: Создание файла и код программы

Код программы:

```
#!/bin/bash
```

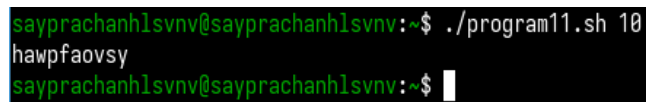
```
a=$1
```

```
letter=(a b c d e f g h i j k l m n o p q r s t u v w x y z)
```



```
for ((i=0; i<a; i++))  
do  
    random=$((RANDOM % 26))  
  
    echo -n "${letter[$random]}"  
done  
echo
```

Я сохраняю файл и запускаю программу, вводя число генерируемых букв в качестве аргумента. В результате программа генерирует 10 случайных букв, то есть программа работает корректно. (рис. 4.9)



```
sayprachanhlsnv@sayprachanhlsnv:~$ ./program11.sh 10  
hawpfaovsy  
sayprachanhlsnv@sayprachanhlsnv:~$
```

Рис. 4.9: Запуск программы

5 Выводы

Во время этой лабораторной работы я изучил основы программирования в оболочке UNIX. Научился писать более сложные пакетные файлы, используя логические структуры управления и циклы.

6 Ответы на контрольные вопросы

1. Найдите синтаксическую ошибку в строке: `while [$1 != "exit"]`

В этой строке есть несколько ошибок. Во-первых, в `bash` после `[` и перед `]` обязательно должны быть пробелы. Без них команда не сработает. Во-вторых, переменную `$1` лучше взять в двойные кавычки, потому что она может содержать пробелы.

Правильная версия будет: `while ["$1" != "exit"]`

2. Как объединить несколько строк в одну?

В `bash` склеить строки можно просто написав их рядом. Например, если есть переменные `VAR1="Hello,"` и `VAR2=" World"`, то `VAR3="$VAR1$VAR2"` создаст строку `Hello, World`. Можно ещё использовать `+=`, например: `VAR1="Hello,"`, потом `VAR1+=" World"`. Результат будет тот же — `Hello, World`.

3. Что делает команда `seq`, и чем её можно заменить в `bash`?

Команда `seq` нужна, чтобы сделать последовательность чисел. Например, `seq 5` выведет от 1 до 5. Можно задать начальное число, шаг и конечное, например: `seq 1 2 9` даст 1, 3, 5, 7, 9. Но эту команду можно заменить обычным циклом в `bash`: `for ((i=1; i<=10; i++))` он делает то же самое. Также можно использовать фигурные скобки: `{1..10}` — это ещё один способ сделать последовательность.

4. Какой результат даст выражение `$((10/3))`?

Ответ будет 3, потому что `bash` делает только целочисленное деление. То есть дробная часть просто отбрасывается. Даже если 10 делится на 3 с остатком, `bash` покажет только целую часть.

5. Чем `zsh` отличается от `bash`?

`Zsh` — это как улучшенная версия `bash`. У неё лучше автодополнение (например, если нажать `Tab` после `cd`, покажет папки), есть встроенный калькулятор `zcalc`, можно работать с числами с запятой. Также поддерживаются хэши (ассоциативные массивы), продвинутый способ дописывания путей, и даже возможность разделить экран терминала, как в `Vim`. В общем, `zsh` удобнее для работы вручную в терминале.

6. Правильна ли конструкция `for ((a=1; a <= LIMIT; a++))`?

Да, конструкция правильная. В `bash` такие циклы с двойными скобками работают как в языке `C`. Там можно не ставить `$` перед переменными. Главное, чтобы переменная `LIMIT` была определена (например, `LIMIT=5`), тогда всё будет работать.

7. Сравните `bash` с другими языками программирования. Какие плюсы и минусы?

`Bash` отлично подходит для автоматизации задач в `Linux` и `macOS`, так как он предустановлен и хорошо интегрируется с системными командами. Он прост в использовании и идеально подходит для работы с файлами и пакетами. Однако это не язык общего назначения, как `Python`, у него нет сложных структур данных, и выполнение команд может быть медленным из-за того, что каждая команда запускается в отдельном процессе. Также `bash`-скрипты не кросс-платформенные и требуют настроек для работы на `Windows`.

Список литературы

Лабораторная работа №14