

Отчет по лабораторной работе №12

Дисциплина: операционные системы

Астраханцева А. А.

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Контрольные вопросы	12
5	Выводы	15

Список иллюстраций

4.1	Создание командного файла №1	8
4.2	Текст программы №1	8
4.3	Проверка работа программы №1	9
4.4	Создание командного файла №2	9
4.5	Содержимое каталога /usr/share/man/man1	10
4.6	Текст программы №2	10
4.7	Вызов командного файла №2	10
4.8	Проверка работа программы №2	11
4.9	Создание командного файла №3	11
4.10	Текст программы №3	11
4.11	Проверка работа программы №3	12

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 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:

1. оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
2. С-оболочка (или csh) — надстройка на оболочкой Борна, использующая С-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
3. 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.

4 Выполнение лабораторной работы

Для начала создадим командный файл №1(рис. 4.1).

```
[aaastrakhantseva@aaastrakhantseva ~]$ touch prog1.sh
[aaastrakhantseva@aaastrakhantseva ~]$ ls
archive  cprog      lab07.sh~  progr2.sh~  Загрузки  Общедоступные
backup   dir1       os-intro   work        Изображения  'Рабочий стол'
bin      lab07.sh   prog1.sh   Документы   Музыка      Шаблоны
[aaastrakhantseva@aaastrakhantseva ~]$
```

Рис. 4.1: Создание командного файла №1

В созданный файл записываем текст нашей программы. Используем оператор while test для получения или запрета доступа(рис. 4.2).

```
Открыть ▾ + prog1.sh
~/

#!/bin/bash

lockfile="./lock.file"
exec {fn}>$lockfile

while test -f "$lockfile"
do
do
if flock -n ${fn}
then
echo "File is blocked"
sleep 5
echo "File is unlocked"
flock -u ${fn}
else
echo "File is blocked"
sleep 5
fi
done
```

Рис. 4.2: Текст программы №1

Проверяем, что работает корректно (рис. 4.3).

```
root@prog1:~/net-ranets-firma-kill-katana$  
[aaastrakhantseva@aaastrakhantseva ~]$ bash prog1.sh  
File is blocked  
File is unlocked  
File is blocked  
File is unlocked  
File is blocked  
File is unlocked  
File is blocked  
File is unlocked  
File is blocked  
File is unlocked  
File is blocked  
File is blocked  
File is unlocked  
File is blocked  
File is unlocked  
File is blocked  
File is unlocked  
File is blocked  
File is unlocked
```

Рис. 4.3: Проверка работа программы №1

Создаем командный файл №2(рис. 4.4).

```
4 [tmux]: 310 недопустимый идентификатор  
[aaastrakhantseva@aaastrakhantseva ~]$ touch prog2.sh  
[aaastrakhantseva@aaastrakhantseva ~]$ ls  
archive  dir1      os-intro  work      Музыка  
backup   lab07.sh  prog1.sh  Документы  Общедоступные  
bin      lab07.sh~ prog2.sh  Загрузки  'Рабочий стол'  
cprog    lock.file progr2.sh~ Изображения  Шаблоны  
[aaastrakhantseva@aaastrakhantseva ~]$
```

Рис. 4.4: Создание командного файла №2

Просматриваем содержимое каталога /usr/share/man/man1 (рис. 4.5).

```
[aaastrakhantseva@aaastrakhantseva ~]$ ls /usr/share/man/man1
.: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
abrt-action-generate-backtrace.1.gz
abrt-action-generate-core-backtrace.1.gz
abrt-action-install-debuginfo.1.gz
abrt-action-list-dsos.1.gz
abrt-action-notify.1.gz
abrt-action-perform-ccpp-analysis.1.gz
abrt-action-save-package-data.1.gz
abrt-action-trim-files.1.gz
abrt-applet.1.gz
abrt-auto-reporting.1.gz
abrt-bodhi.1.gz
abrt-cli.1.gz
```

Рис. 4.5: Содержимое каталога /usr/share/man/man1

В командный файл записываем текст программы. Используем оператор test для проверки того, есть ли в данном каталоге справка для запрошенной команды(рис. 4.6).

```
#!/bin/bash
a=$1
if test -f "/usr/share/man/man1/${a}.1.gz"
then less /usr/share/man/man1/${a}.1.gz
else
echo "There is no such command"
fi
```

Рис. 4.6: Текст программы №2

Проверяем, что работает корректно (рис. 4.7 - 4.8).

```
[aaastrakhantseva@aaastrakhantseva ~]$ bash prog2.sh pwd
[aaastrakhantseva@aaastrakhantseva ~]$
```

Рис. 4.7: Вызов командного файла №2

```
aastrakhantseva@aastrakhantseva:~$ bash prog2.sh pwd
PWD(1)
User Commands
PWD(1)
ESC [mNAMEESC [Om
pwd - print name of current/working directory
ESC [mSYNOPSISESC [Om
ESC [mpwd ESC [22mESC [4mOPTIONESC [24m]...
ESC [mDESCRIPTIONESC [Om
Print the full filename of the current working directory.
ESC [Im-ESC [22m, ESC [Im--logicalESC [Om
use PWD from environment, even if it contains symlinks
ESC [Im-PESC [22m, ESC [Im--physicalESC [Om
avoid all symlinks
ESC [Im--helpESC [22mdisplay this help and exit
ESC [Im--versionESC [Om
output version information and exit
If no option is specified, ESC [Im-P ESC [22mis assumed.
NOTE: your shell may have its own version of pwd, which usually super-
sedes the version described here. Please refer to your shell's docu-
mentation for details about the options it supports.
ESC [mAUTHORESC [Om
Written by Jim Meyering.
```

Рис. 4.8: Проверка работа программы №2

Создаем командный файл №3 (рис. 4.9).

```
[aastrakhantseva@aastrakhantseva ~]$ touch prog3.sh
[aastrakhantseva@aastrakhantseva ~]$ ls
archive  dir1      os-intro  progr2.sh  Изображения  Шаблоны
backup   lab07.sh  prog1.sh  work       Музыка
bin      lab07.sh~ prog2.sh  Документы  Общедоступные
cprog    lock.file prog3.sh  Загрузки   'Рабочий стол'
```

Рис. 4.9: Создание командного файла №3

В командный файл записываем текст программы. Используем цикл for и оператор case(рис. 4.10).

```
#!/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 r;; 18) echo -n s;;
    19) echo -n t;; 20) echo -n q;; 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. 4.10: Текст программы №3

Проверяем, что работает корректно (рис. 4.11).

```
[aastrakhantseva@aastrakhantseva ~]$ bash prog3.sh 14
osnxntfoixgeqn
[aastrakhantseva@aastrakhantseva ~]$
```

Рис. 4.11: Проверка работа программы №3

4.1 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `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. Как объединить (конкатенация) несколько строк в одну?

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. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать её функционал при программировании на `bash`?

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. Какой результат даст вычисление выражения $\$((10/3))$?

Результатом данного выражения $\$((10/3))$ будет 3, потому что это целочисленное деление без остатка.

5. Укажите кратко основные отличия командной оболочки zsh от bash.

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. Проверьте, верен ли синтаксис данной конструкции `for ((a=1; a <= LIMIT; a++))`

`for ((a=1; a <= LIMIT; a++))` синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать \$ перед переменными ().

7. Сравните язык bash с какими-либо языками программирования. Какие преимущества у bash по сравнению с ними? Какие недостатки?

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: 1. Дополнительные библиотеки других языков позволяют выполнить больше действий 2. Bash не является языком общего назначения 3. Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на скорости выполнения этого скрипта 4. Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий

5 Выводы

В ходе выполнения лабораторной работы №12 я изучила основы программирования в оболочке ОС UNIX и научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.