

Отчет по лабораторной работе №11

Программирование в командном процессоре ОС UNIX.
Расширенное программирование

Сомских Даниил Сергеевич

Содержание

1	Цель работы	4
2	Задание	5
3	Выполнение лабораторной работы	7
4	Контрольные вопросы	13
5	Выводы	18
	Список литературы	19

Список иллюстраций

3.1 Командный файл №1	8
3.2 Работа кода №1	8
3.3 Каталог man	9
3.4 Командный файл №2	10
3.5 Работа программы №2	10
3.6 Справка команды ls	11
3.7 Командный файл №3	11
3.8 Результат выполнения командного файла №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 semaфоров. 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. Напишем командный файл, реализующий упрощённый механизм semaфоров. 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. (рис. 3.1):

содержимое каталога /usr/share/man/man1 (рис. 3.3). 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 получает в виде аргумента командной строки название команды и в виде результата выдаёт справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1. (рис. 3.4):

```
ussomskrov@ussomskrov:~$ 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-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-list-dsos.1.gz
abrt-action-notify.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
abrt-dump-journal-core.1.gz
abrt-dump-journal-oops.1.gz
abrt-dump-journal-xorg.1.gz
abrt-dump-oops.1.gz
abrt-dump-xorg.1.gz
abrt-handle-upload.1.gz
abrt-harvest-pstoreoops.1.gz
abrt-harvest-vmcore.1.gz
abrt-merge-pstoreoops.1.gz
abrt-server.1.gz
abrt-watch-log.1.gz
ac.1.gz
ndctl-start-scrub.1.gz
ndctl-update-firmware.1.gz
ndctl-update-passphrase.1.gz
ndctl-wait-overwrite.1.gz
ndctl-wait-scrub.1.gz
ndctl-write-infoblock.1.gz
ndctl-write-labels.1.gz
ndctl-zero-labels.1.gz
neato.1.gz
neqn.1.gz
netavark.1.gz
netkey-tool.1.gz
networkctl.1.gz
newgidmap.1.gz
newgrp.1.gz
newuidmap.1.gz
ngettext.1.gz
nhlt-dmic-info.1.gz
nice.1.gz
nisdomainname.1.gz
nl.1.gz
nm.1.gz
nmblookup.1.gz
nmcli.1.gz
nm-connection-editor.1.gz
nm-online.1.gz
nohup.1.gz
nop.1.gz
notify-send.1.gz
nproc.1.gz
nroff.1.gz
nsec3hash.1.gz
nsenter.1.gz
nseq.1.gz
nsslookup.1.gz
nsupdate.1.gz
numfmt.1.gz
nvme.1.gz
```

Рис. 3.3: Каталог man

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

Рис. 3.4: Командный файл №2

Делаем файлы исполняемыми и проверяем работу программы, запросив справку о команде `ls` (рис. 3.5)

```
dssomsikov@dssomsikov:~$ gedit prog11-2.sh
dssomsikov@dssomsikov:~$ chmod +x prog11-2.sh
dssomsikov@dssomsikov:~$ bash prog11-2.sh
There is no such command
dssomsikov@dssomsikov:~$ bash prog11-2.sh ls
```

Рис. 3.5: Работа программы №2

В итоге мы получаем справку команды `ls`, которую запрашивали (рис. 3.6).

```
ESC[4mL$ESC[24m(1) User Commands
ESC[4mL$ESC[24m(1)

ESC[1mNAMEESC[0m
ls - list directory contents

ESC[1mSYNOPSISESC[0m
ESC[1mls ESC[22m[ESC[4mOPTIONESC[24m]... [ESC[4mFILEESC[24m]...

ESC[1mDESCRIPTIONESC[0m
List information about the FILES (the current directory by default). Sort entries alphabetically if none
of ESC[1m-cftuvSUX ESC[22mnor ESC[1m--sort ESC[22mis specified.

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

ESC[1m-aESC[22m, ESC[1m--allESC[0m
do not ignore entries starting with .

ESC[1m-AESC[22m, ESC[1m--almost-allESC[0m
do not list implied . and ..

ESC[1m--authorESC[0m
with ESC[1m-lESC[22m, print the author of each file

ESC[1m-bESC[22m, ESC[1m--escapeESC[0m
print C-style escapes for nongraphic characters

ESC[1m--block-sizeESC[22m=ESC[4mSIZEESC[0m
with ESC[1m-lESC[22m, scale sizes by SIZE when printing them; e.g., '--block-size=M'; see SIZE format below

ESC[1m-BESC[22m, ESC[1m--ignore-backupsESC[0m
do not list implied entries ending with ~

ESC[1m-c ESC[22mwith ESC[1m-ltESC[22m: sort by, and show, ctime (time of last change of file status information); with ESC[1m-lESC[22m: show
ctime and sort by name; otherwise: sort by ctime, newest first
```

Рис. 3.6: Справка команды `ls`

3. Используя встроенную переменную `$RANDOM`, напомним командный файл, генерирующий случайную последовательность букв латинского алфавита (рис. fig. 3.7). Учтём, что `$RANDOM` выдаёт псевдослучайные числа в диапазоне от 0 до 32767 (рис. 3.7):

```
1 #!/bin/bash
2 a=$1
3 for ((i=0; i<$a; i++))
4 do
5     ((char=$RANDOM%26+1))
6     case $char in
7         1) echo -n a;; 2) echo -n b;; 3) echo -n c;; 4) echo -n d;; 5) echo -n e;;
8         6) echo -n f;; 7) echo -n g;; 8) echo -n h;; 9) echo -n i;; 10) echo -n j;;
9         11) echo -n k;; 12) echo -n l;; 13) echo -n m;; 14) echo -n n;; 15) echo -n o;;
10        16) echo -n p;; 17) echo -n r;; 18) echo -n s;; 19) echo -n t;; 20) echo -n q;;
11        21) echo -n u;; 22) echo -n v;; 23) echo -n w;; 24) echo -n x;; 25) echo -n y;; 26) echo -n z;;
12    esac
13 done
14 echo
```

Рис. 3.7: Командный файл №3

Делаем файлы исполняемыми и выводим результат (рис. 3.8).

```
dssomsikov@dssomsikov:~$ gedit prog11-3.sh
dssomsikov@dssomsikov:~$ chmod +x prog11-3.sh
dssomsikov@dssomsikov:~$ bash prog11-3.sh 10
zzkpwpozyn
dssomsikov@dssomsikov:~$ bash prog11-3.sh 10
vddxvzodbw
dssomsikov@dssomsikov:~$ bash prog11-3.sh 100
aybyxskcquvmkuqdzqmvjmqmaofzjeeppjuybgjvqxlrlegxfhssvzsliffkpefuonjescbzunwhmbaxmvciplyiuzcsrnpb
dssomsikov@dssomsikov:~$
```

Рис. 3.8: Результат выполнения командного файла №3

4 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `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:

```
bash while [ "$1" != "exit" ]
```

2. Как объединить (конкатенация) несколько строк в одну?

Для объединения нескольких строк в одну в `bash`-скриптах можно использовать следующие методы:

- Просто написать строки одну за другой без разделителей:

```
bash str="Строка1" "Строка2" "Строка3"
```

- Использовать оператор конкатенации `(+=)`, чтобы добавить к строке дополнительные данные:

```
bash str="Строка1" str+="Строка2" str+="Строка3"
```

- Использовать команду `printf` для форматирования и объединения строк:

```
bash printf -v str "%s%s%s" "Строка1" "Строка2" "Строка3"
```

- Использовать команду `echo` с опцией `-n` для предотвращения добавления новой строки после каждого вызова:

```
bash str=$(echo -n "Строка1"; echo -n "Строка2"; echo -n "Стро-
ка3")
```

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

Утилита `seq` в `bash` используется для генерации последовательности чисел. Она позволяет указать начальное число, шаг и конечное число для создания последовательности. Например, команда `seq 1 2 10` выведет числа от 1 до 10 с шагом 2. Чтобы реализовать функционал `seq` без использования самой утилиты, можно применить следующие подходы:

- Использование цикла `for`: `bash for ((i=1; i<=10; i+=2)); do echo $i done`
- Использование цикла `while`: `bash i=1 while [$i -le 10]; do echo ii=$((i+2)) done`
- Использование `brace expansion` `{}`: `bash echo {1..10..2}`

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

Данное выражение в `bash` приведёт к выполнению целочисленного деления, так как `bash` не поддерживает вещественную арифметику в арифметических операциях. В результате, вы получите целую часть от деления 10 на 3, то есть 3.

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

Командная оболочка `zsh` часто сравнивается с `bash`, так как обе они широко используются и имеют много общего. Однако между ними есть ряд отличий:

- **Интерактивность:** `zsh` предлагает более продвинутые возможности для интерактивной работы, включая улучшенное автодополнение и коррекцию ошибок.
- **Темы и плагины:** `zsh` поддерживает темы и плагины через фреймворк `oh-my-zsh`, что позволяет легко настраивать внешний вид и функциональность оболочки.
- **Синтаксис:** В `zsh` есть улучшения синтаксиса, такие как более гибкие глобальные выражения и расширенные возможности для работы со строками и массивами.
- **Совместимость:** `zsh` обычно совместим с `bash`, но включает дополнительные функции, которые могут не работать в `bash`.
- **Конфигурация:** Файлы конфигурации для `zsh` (`~/.zshrc`) и `bash` (`~/.bashrc` и `~/.bash_profile`) различаются, и `zsh` предоставляет более обширные настройки по умолчанию.

Эти отличия делают `zsh` популярным выбором для пользователей, которые ищут более богатый пользовательский интерфейс и гибкость в настройке своей командной оболочки. Однако `bash` остаётся стандартом для многих систем и скриптов из-за его широкой доступности и предсказуемости.

6. Проверьте, верен ли синтаксис данной конструкции `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?

Сравнение `bash` с другими языками программирования можно провести по нескольким критериям:

Преимущества `bash`:

- Встроенная поддержка: Практически в каждой Unix-подобной системе есть `bash`, что делает его универсальным инструментом для системного администрирования.
- Интеграция с системой: `bash` обладает прямым доступом к системным вызовам и утилитам командной строки.
- Простота использования: Для написания простых скриптов не требуется сложная настройка или компиляция.

Недостатки `bash`:

- Ограниченная область применения: `bash` не подходит для сложных программных систем, где требуется высокая производительность и масштабируемость.
- Отсутствие современных функций: В `bash` нет многих возможностей, доступных в полноценных языках программирования, таких как объектно-ориентированное программирование.
- Сложность: Сложные скрипты на `bash` могут быть трудночитаемыми и поддерживаемыми из-за ограничений синтаксиса и структуры языка.

Сравнивая `bash` с языками программирования, такими как Python или Java, можно отметить, что эти языки предлагают более широкие возможности для разработки программного обеспечения, включая библиотеки для научных расчетов, машинного обучения, веб-

разработки и многого другого. Однако для системного администрирования и быстрой автоматизации задач `bash` остается предпочтительным выбором из-за его простоты и тесной интеграции с операционной системой.

5 Выводы

В данной лабораторной работе мы изучили основы программирования в оболочке ОС UNIX и научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Список литературы

1. Руководство к лабораторной работе №11.