

Отчет по лабораторной работе №12

Дисциплина: Операционные системы

Шошина Евгения Александровна (НКАбд-03-22)

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	8
4	Выполнение лабораторной работы	9
5	Выводы	13
6	Контрольные вопросы	14

Список иллюстраций

4.1	Рис. 1.12: Gedit prog1.sh	9
4.2	Рис. 2.12: Текст первой программы	10
4.3	Рис. 3.12: Результат	10
4.4	Рис. 4.12: Gedit prog2.sh	11
4.5	Рис. 5.12: Текст второй программы	11
4.6	Рис. 6.12 Результат	11
4.7	Рис. 7.12: Gedit prog3.sh	12
4.8	Рис. 8.12: Текст третьей программы	12
4.9	Рис. 9.12: Результат	12

Список таблиц

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

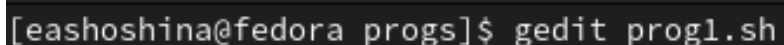
32767.

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: – 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 [1]


4 Выполнение лабораторной работы

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. (рис. 4.2, 4.3)



```
[eashoshina@fedora progs]$ gedit prog1.sh
```

Рис. 4.1: Рис. 1.12: Gedit prog1.sh



```
1 #!/bin/bash
2 lockfile="./lock.file"
3 exec {fn}>${lockfile}
4 while test -f "${lockfile}"
5 do
6 if flock -n ${fn}
7 then
8 echo "File is blocked"
9 sleep 5
10 echo "File is unlocked"
11 flock -u ${fn}
12 else
13 echo "File is blocked"
14 sleep 5
15 fi
16 done
```

Рис. 4.2: Рис. 2.12: Текст первой программы

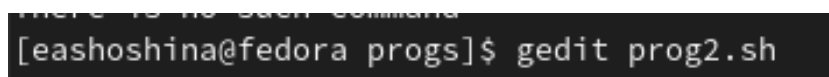


```
[eashoshina@fedora progs]$ bash prog1.sh
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
```

Рис. 4.3: Рис. 3.12: Результат

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`. (рис. 4.5, 4.7, 4.6)



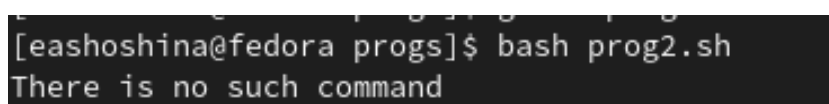
```
[eashoshina@fedora progs]$ gedit prog2.sh
```

Рис. 4.4: Рис. 4.12: Gedit prog2.sh



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

Рис. 4.5: Рис. 5.12: Текст второй программы



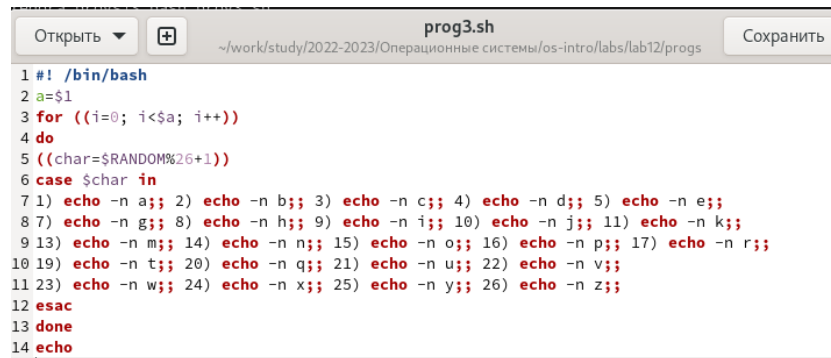
```
[eashoshina@fedora progs]$ bash prog2.sh
There is no such command
```

Рис. 4.6: Рис. 6.12 Результат

- 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. (рис. 4.8, 4.9)

```
[eashoshina@fedora progs]$ gedit prog3.sh
```

Рис. 4.7: Рис. 7.12: Gedit prog3.sh



```
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 7) echo -n g;; 8) echo -n h;; 9) echo -n i;; 10) echo -n j;; 11) echo -n k;;
9 13) echo -n m;; 14) echo -n n;; 15) echo -n o;; 16) echo -n p;; 17) echo -n r;;
10 19) echo -n t;; 20) echo -n q;; 21) echo -n u;; 22) echo -n v;;
11 23) echo -n w;; 24) echo -n x;; 25) echo -n y;; 26) echo -n z;;
12 esac
13 done
14 echo
```

Рис. 4.8: Рис. 8.12: Текст третьей программы

```
[eashoshina@fedora progs]$ bash prog3.sh 25
tajwrchbcjmtvuebtmwettw
```

Рис. 4.9: Рис. 9.12: Результат

5 Выводы

В процессе выполнения этой лабораторной работы я продолжила осваивать программирование на bash.

6 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 `"`, по- тому 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="␣␣1VAR2" echo "␣ ␣␣3" ␣ ␣␣␣␣␣, ␣ ␣␣␣␣ ␣ ␣ ␣␣1 = "␣␣␣␣␣,"␣ ␣␣1+= "␣␣␣␣␣"␣␣␣␣␣"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++))`
 - 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 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.