

Лабораторная работа №12

**Программирование в командном процессоре ОС UNIX. Расширенное
программирование**

Сиссе Мохамед Ламин;НММбд-01-22

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Выводы	13
6	Контрольные вопросы	14
	Список литературы	17

Список иллюстраций

4.1	Текст первой программы	8
4.2	Результат	9
4.3	Текст второй программы	10
4.4	Результат	10
4.5	Результат	11
4.6	Текст третьей программы	11
4.7	Результат	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

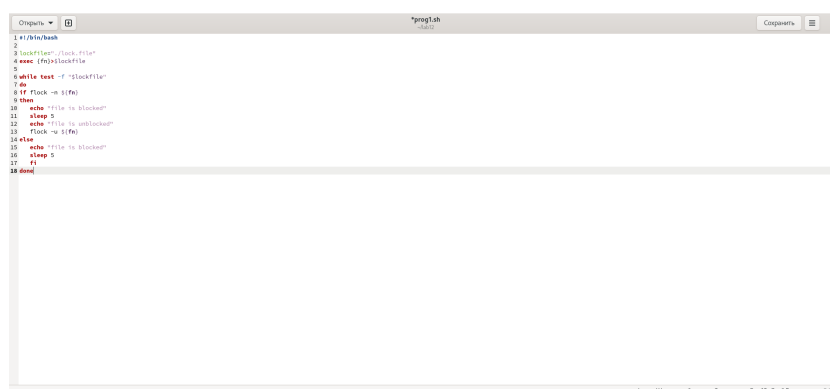
32767.

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: – 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 оболочке 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 [Prog:bash?].

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. [4.1])



```
1 #!/bin/bash
2
3 lockfile="/lock_file"
4 exec (flock -w $lockfile)
5
6 while test -f "$lockfile"
7 do
8     if flock -w $lockfile
9     then
10         echo "file is blocked"
11         sleep 5
12         echo "file is unblocked"
13         flock -w $lockfile
14     else
15         echo "file is blocked"
16         sleep 5
17     fi
18 done
```

Рис. 4.1: Текст первой программы


```
mohamedlamine@fedora:~/lab12
[mohamedlamine@fedora ~]$ cd lab12
bash: cd: lab12: Нет такого файла или каталога
[mohamedlamine@fedora ~]$ mkdir lab12
[mohamedlamine@fedora ~]$ ls
arch-pc      equiplist2  lab07.sh    pandoc      reports     Загрузки
australia   feathers    lab10-3     parentdir   ski.plases  Изображения
backup      file.txt    lab11       play        text.txt    Музыка
conf.txt    fun         lab12       prog1.sh    tutorial     Общедоступные
dir1        lab04       may         prog2.sh    work        'Рабочий стол'
dir2        '##lab07.sh##'  monthly    prog3.sh    Видео
dir3        '#lab07.sh#'  my_os      prog4.sh    Документы

[mohamedlamine@fedora ~]$ cd lab12
[mohamedlamine@fedora lab12]$ touch prog1.sh
[mohamedlamine@fedora lab12]$ bash prog1.sh
file is blocked
file is unblocked
file is blocked
file is unblocked
file is blocked
^C
[mohamedlamine@fedora lab12]$
```

Рис. 4.2: 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. [4.3])

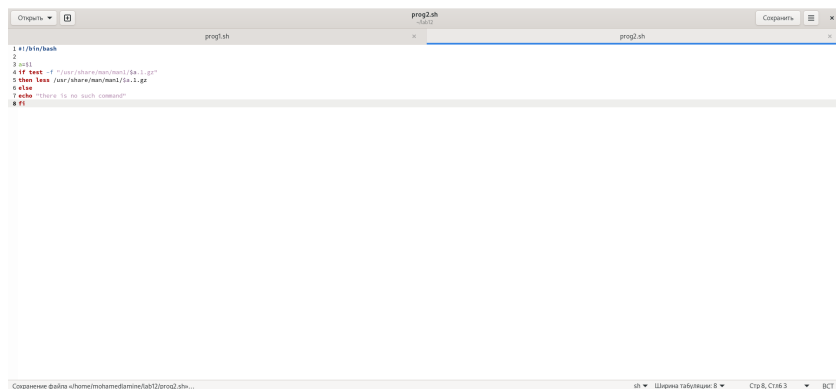


Рис. 4.3: Текст второй программы

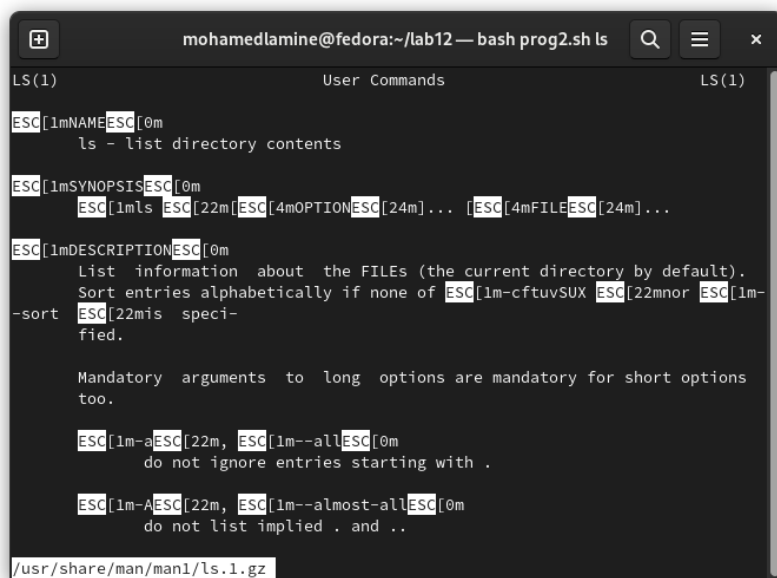
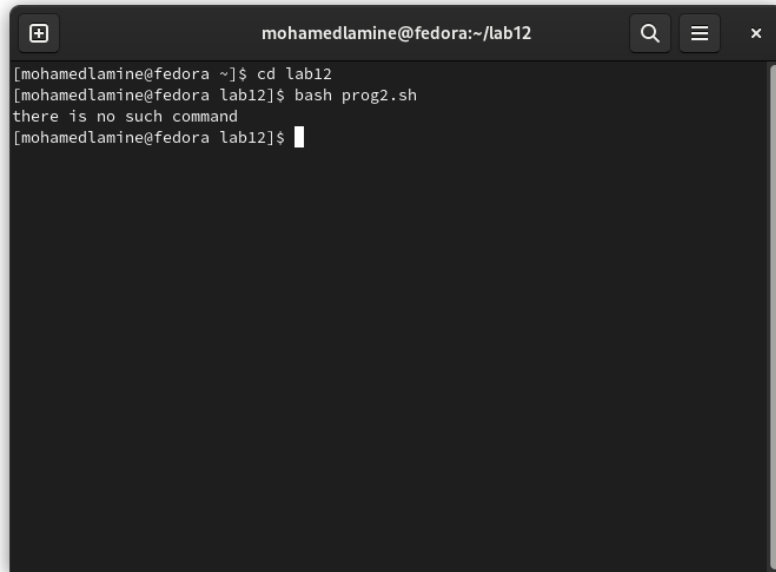


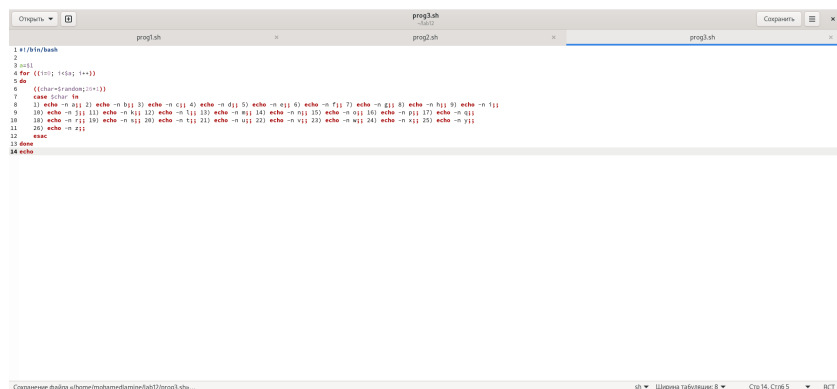
Рис. 4.4: Результат



```
mohamedlamine@fedora: ~/lab12
[mohamedlamine@fedora ~]$ cd lab12
[mohamedlamine@fedora lab12]$ bash prog2.sh
there is no such command
[mohamedlamine@fedora lab12]$
```

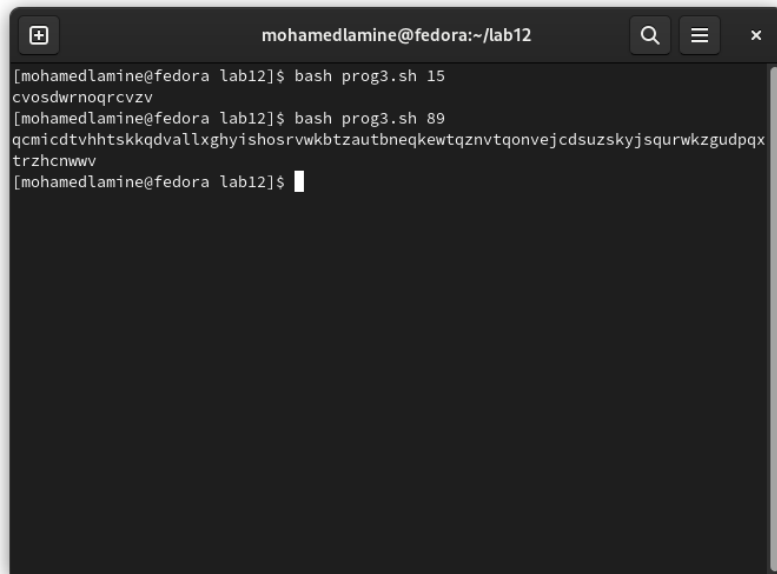
Рис. 4.5: Результат

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. (рис. [4.6])



```
1 #!/bin/bash
2
3 #
4 for ((i=0; i<50; i++))
5 do
6     ((char=$((RANDOM%26)))>0)
7     case $char in
8         0) echo -n a;; 1) echo -n b;; 2) echo -n c;; 3) echo -n d;; 4) echo -n e;; 5) echo -n f;; 6) echo -n g;; 7) echo -n h;; 8) echo -n i;;
9         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 q;;
10        18) echo -n r;; 19) echo -n s;; 20) echo -n t;; 21) echo -n u;; 22) echo -n v;; 23) echo -n w;; 24) echo -n x;; 25) echo -n y;;
11        26) echo -n z;;
12    esac
13 done
14 echo
```

Рис. 4.6: Текст третьей программы



A terminal window titled "mohamedlamine@fedora:~/lab12" with search, menu, and close icons. The terminal shows the execution of a script "prog3.sh" with arguments "15" and "89". The output for "15" is "cvosdwrnoqrcvzv" and for "89" is a long alphanumeric string. The prompt returns to the user after each execution.

```
mohamedlamine@fedora:~/lab12
[mohamedlamine@fedora lab12]$ bash prog3.sh 15
cvosdwrnoqrcvzv
[mohamedlamine@fedora lab12]$ bash prog3.sh 89
qcmicdtvhhtskkqdvallxghyishosrvwkbztzautbneqkewtqznvtqonvejcdsuzskyjsqurwkzgudpqx
trzhcnwvv
[mohamedlamine@fedora lab12]$
```

Рис. 4.7: Результат

5 Выводы

В процессе выполнения этой лабораторной работы я продолжил 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 `"`, 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 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

Список литературы