

Отчёт по лабораторной работе 12

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

Сидорова Наталья Андреевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	8
4	Выполнение лабораторной работы	9
5	Выводы	14
	Список литературы	15

Список иллюстраций

4.1	Код к первому заданию	9
4.2	Работа первого кода	10
4.3	Код ко второму заданию	10
4.4	Работа второго кода	11
4.5	Код к третьему заданию	11
4.6	Работа третьего кода	11

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

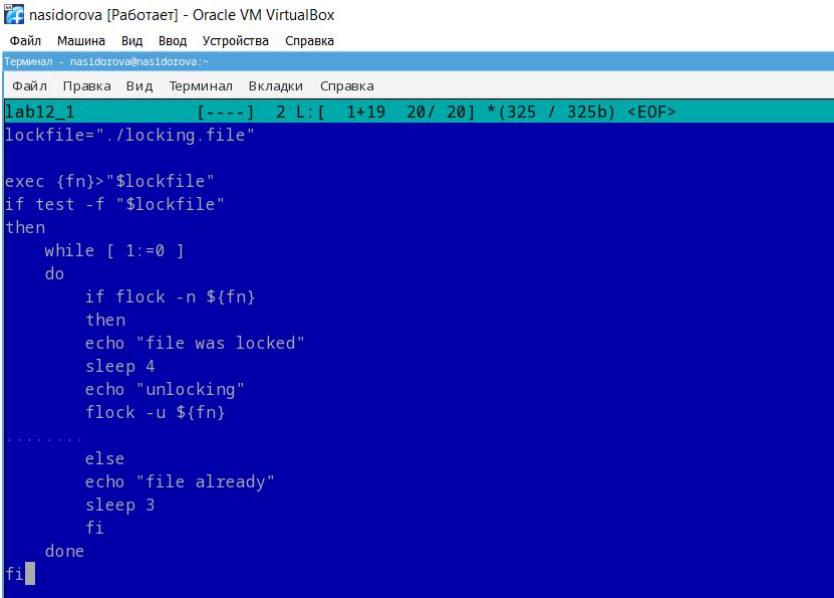
32767.

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; – 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.

4 Выполнение лабораторной работы

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 имела возможность взаимодействия трёх и более процессов. (рис. 4.1).



```
nasidorova [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
терминал - nasidorova@nasidorova:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
lab12_1 [----] 2 L: [ 1+19 20/ 20] *(325 / 325b) <EOF>
lockfile="/locking.file"

exec {fn}>"$lockfile"
if test -f "$lockfile"
then
  while [ 1:=0 ]
  do
    if flock -n ${fn}
    then
      echo "file was locked"
      sleep 4
      echo "unlocking"
      flock -u ${fn}
    .....
    else
      echo "file already"
      sleep 3
    fi
  done
fi
```

Рис. 4.1: Код к первому заданию

Запуск программы 1 (рис. 4.2).

```
[nasidorova@nasidorova ~]$ mcedit lab12_1

[nasidorova@nasidorova ~]$ chmod +x lab12_1
[nasidorova@nasidorova ~]$ ./lab12_1
file was locked
unlocking
file was locked
unlocking
file was locked
unlocking
file was locked
```

Рис. 4.2: Работа первого кода

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).

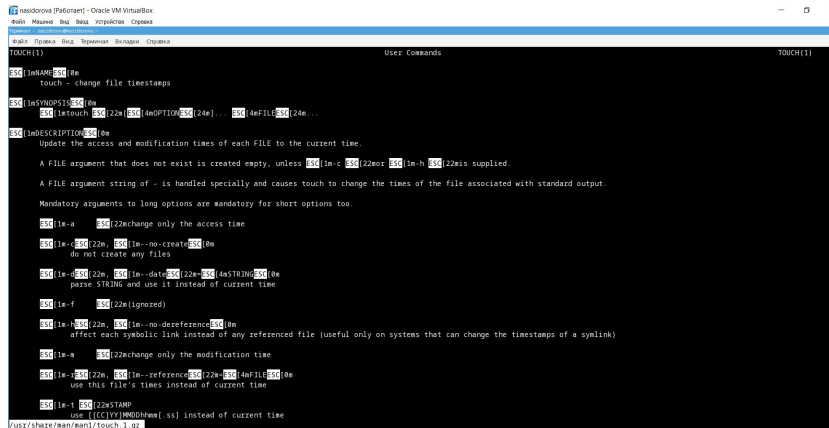
```
nasidorova [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
Терминал - nasidorova@nasidorova:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
lab12_2  [-----]  2 L: [ 1+13 14/ 14] *(204 / 204b) <EOF>
command=""

while getopts :n: opt
do
case $opt in
n)command="$OPTARG";;
esac
done

if test -f "/usr/share/man/man1/$command.1.gz"
then less /usr/share/man/man1/$command.1.gz
else
echo "No such command"
fi
```

Рис. 4.3: Код ко второму заданию

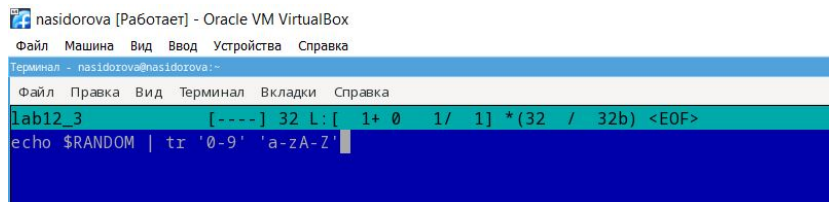
Запуск программы 2 (рис. 4.4).



```
nasidorova [Работает] - Oracle VM VirtualBox
Файл Машина Вид Ввод Устройства Справка
Терминал - nasidorova@nasidorova:~
Файл Правка Вид Терминал Вкладки Справка
lab12_3 [----] 32 L: [ 1+ 0 1/ 1] *(32 / 32b) <EOF>
echo $RANDOM | tr '0-9' 'a-zA-Z'
```

Рис. 4.4: Работа второго кода

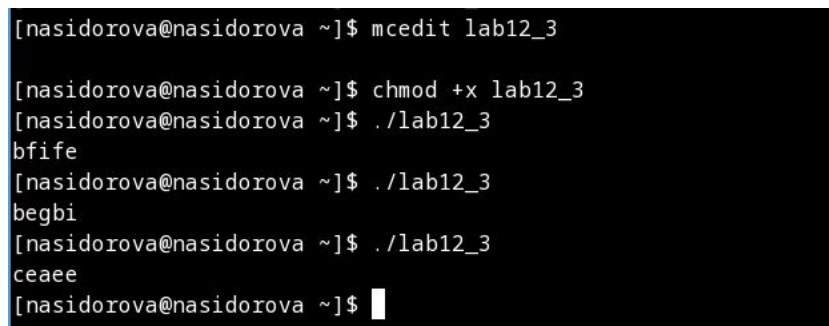
Используя встроенную переменную \$RANDOM, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767. (рис. 4.5).



```
nasidorova [Работает] - Oracle VM VirtualBox
Файл Машина Вид Ввод Устройства Справка
Терминал - nasidorova@nasidorova:~
Файл Правка Вид Терминал Вкладки Справка
lab12_3 [----] 32 L: [ 1+ 0 1/ 1] *(32 / 32b) <EOF>
echo $RANDOM | tr '0-9' 'a-zA-Z'
```

Рис. 4.5: Код к третьему заданию

Запуск программы 3 (рис. 4.4).



```
[nasidorova@nasidorova ~]$ mcedit lab12_3
[nasidorova@nasidorova ~]$ chmod +x lab12_3
[nasidorova@nasidorova ~]$ ./lab12_3
bfife
[nasidorova@nasidorova ~]$ ./lab12_3
begbi
[nasidorova@nasidorova ~]$ ./lab12_3
ceae
[nasidorova@nasidorova ~]$
```

Рис. 4.6: Работа третьего кода

Контрольные вопросы: 1. Найдите синтаксическую ошибку в следующей строке: `1 while [$1 != "exit"]` В данной строке 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 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

5 Выводы

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

Список литературы