

Лабораторная работа №12.

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

Александр Андреевич Шуплецов

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение работы	7
4	Контрольные вопросы	10
5	Выводы	13
	Список литературы	14

Список иллюстраций

3.1	текст командного файла, реализующего упрощенный механизм семафоров	7
3.2	проверка командного файла, реализующего упрощённый механизм семафоров	8
3.3	текст командного файла, реализующего команду tap	8
3.4	проверка командного файла, реализующего команду tap	8
3.5	текст командного файла, генерирующего случайную последовательность букв	9
3.6	проверка командного файла, генерирующего случайную последовательность букв	9

Список таблиц

1 Цель работы

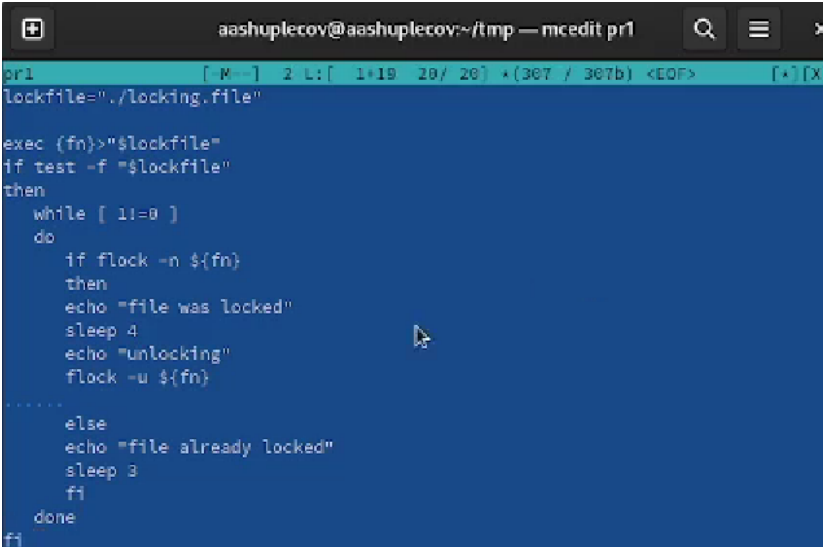
Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 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.

3 Выполнение работы

1. Напишем командный файл, реализующий упрощённый механизм семафоров.

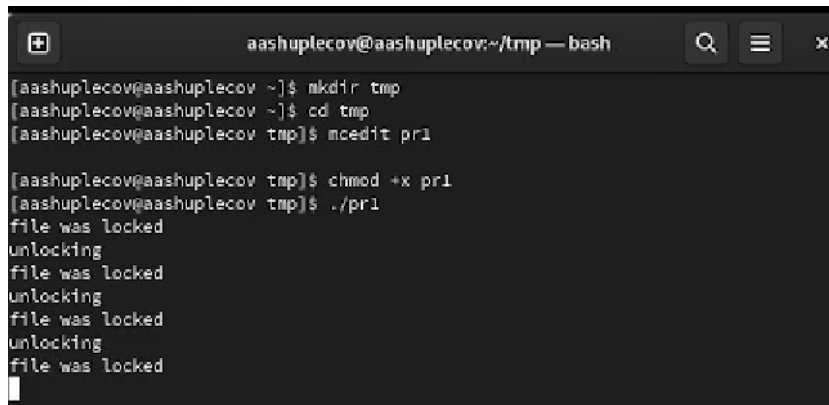
A screenshot of a terminal window with a dark blue background. The window title is 'aashuplecov@aashuplecov:~/tmp — mcedit pr1'. The script content is as follows:

```
pr1
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 locked"
      sleep 3
    fi
  done
fi
```

Рис. 3.1: текст командного файла, реализующего упрощенный механизм семафоров

2. Убедимся, что командный файл, реализующий упрощённый механизм семафоров, работает.



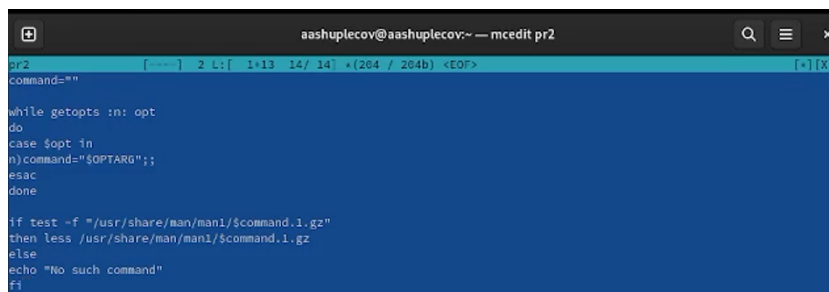
```
aashuplecov@aashuplecov:~/tmp — bash
[aashuplecov@aashuplecov ~]$ mkdir tmp
[aashuplecov@aashuplecov ~]$ cd tmp
[aashuplecov@aashuplecov tmp]$ mcedit pr1

[aashuplecov@aashuplecov tmp]$ chmod +x pr1
[aashuplecov@aashuplecov tmp]$ ./pr1
file was locked
unlocking
file was locked
unlocking
file was locked
unlocking
file was locked
unlocking
file was locked

```

Рис. 3.2: проверка командного файла, реализующего упрощённый механизм семафоров

3. Напишем командный файл, реализующий команду man.



```
pr2 [-----] 2 L: [ 1+13 14/ 14] +(264 / 264b) <EOF> [+][X]
command=""

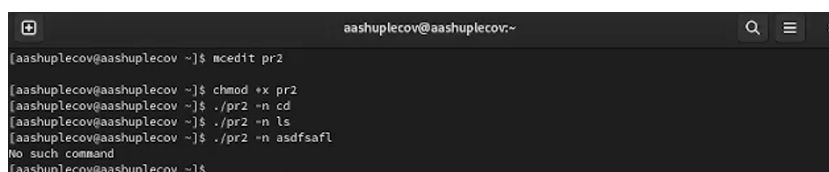
while getopts in: opt
do
case $opt in
n)command="$OPTARG";;
esac
done

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. 3.3: текст командного файла, реализующего команду man

4. Убедимся, что командный файл, реализующий команду man, работает.



```
aashuplecov@aashuplecov:~$ mcedit pr2

[aashuplecov@aashuplecov ~]$ chmod +x pr2
[aashuplecov@aashuplecov ~]$ ./pr2 -n cd
[aashuplecov@aashuplecov ~]$ ./pr2 -n ls
[aashuplecov@aashuplecov ~]$ ./pr2 -n asdfsaf1
No such command
[aashuplecov@aashuplecov ~]$

```

Рис. 3.4: проверка командного файла, реализующего команду man

5. Используя встроенную переменную \$RANDOM, напишем командный файл, генерирующий случайную последовательность букв латинского алфавита.

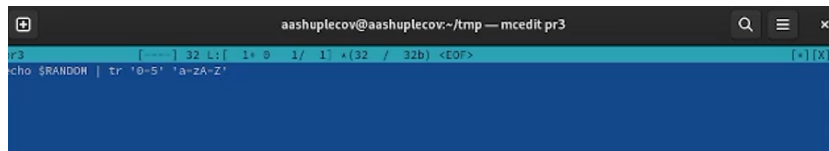


Рис. 3.5: текст командного файла, генерирующего случайную последовательность букв

6. Убедимся, что командный файл, генирирующий случайную последовательность букв латинского алфавита, работает.

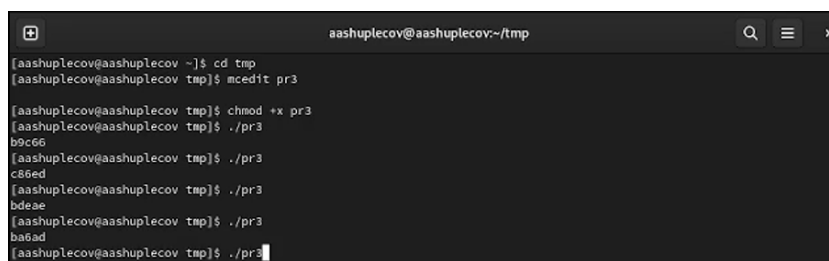


Рис. 3.6: проверка командного файла, генерирующего случайную последовательность букв

4 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

```
1 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 дистрибутивов Linux
- * 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, могут вызвать зависимость от других утилит
- * Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных усилий

5 Выводы

Я изучил основы программирования в оболочке ОС UNIX, научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Список литературы

Кулябов Д.С. “Материалы к лабораторным работам”