

# **Отчет по лабораторной работе №12**

**Операционные системы**

Шуваев Сергей Александрович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>8</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>10</b>
<b>5</b>	<b>Выводы</b>	<b>15</b>
<b>6</b>	<b>Ответы на контрольные вопросы</b>	<b>16</b>

## Список иллюстраций

4.1	Создаю командный файл для первой программы, пишу ее, проверяю ее работу . . . . .	10
4.2	Командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени t2<>t1, 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 . . . . .	11
4.3	Чтобы реализовать команду man с помощью командного файла, изучаю 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 . . . . .	12
4.4	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 . . . . .	12
4.5	Проверяю работу командного файла . . . . .	13
4.6	Командный файл работает так же, как и команда man, открывает справку по указанной утилите . . . . .	13
4.7	Создаю файл для кода третьей программы, пишу программу и проверяю ее работу . . . . .	13
4.8	Используя встроенную переменную \$RANDOM, пишу командный файл, генерирующий случайную последовательность букв латинского алфавита. Т.к. \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767, ввожу ограничения так, чтобы была генерация чисел от 1 до 26 . . . . .	14

## Список таблиц

# 1 Цель работы

Цель данной лабораторной 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).

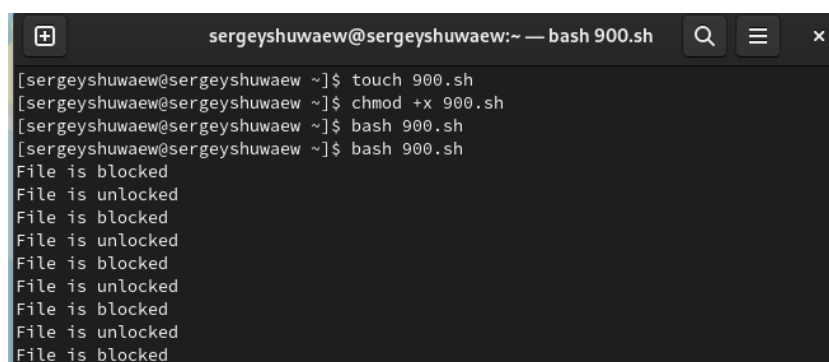
POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с



описанными ниже.


## 4 Выполнение лабораторной работы

Описываются проведённые действия, в качестве иллюстрации даётся ссылка на иллюстрацию (рис. 4.8).



```
sergeyshuwaew@sergeyshuwaew:~ — bash 900.sh
[sergeyshuwaew@sergeyshuwaew ~]$ touch 900.sh
[sergeyshuwaew@sergeyshuwaew ~]$ chmod +x 900.sh
[sergeyshuwaew@sergeyshuwaew ~]$ bash 900.sh
[sergeyshuwaew@sergeyshuwaew ~]$ bash 900.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
```

Рис. 4.1: Создаю командный файл для первой программы, пишу ее, проверяю ее работу



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

Рис. 4.2: 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

```
[sergeyshuwaew@sergeyshuwaew ~]$ 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-core.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-install-debuginfo.1.gz
abrt-action-list-dsos.1.gz
abrt-action-notify.1.gz
abrt-action-perform-ccpp-analysis.1.gz
```

Рис. 4.3: Чтобы реализовать команду man с помощью командного файла, изучаю 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



```
Открыть ▾ 901.sh Стр. 8, Поз. 3
● report.md 901.sh x
1 #!/bin/bash
2
3 a=$1
4 if test -f "/usr/share/man/man1/${a}.1.gz"
5 then less /usr/share/man/man1/${a}.1.gz
6 else
7 echo "There is no such command"
8 fi
```

Рис. 4.4: 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

```
[sergeyshuwaew@sergeyshuwaew ~]$ touch 901.sh
[sergeyshuwaew@sergeyshuwaew ~]$ chmod +x 901.sh
[sergeyshuwaew@sergeyshuwaew ~]$ ./901.sh ls
```

Рис. 4.5: Проверяю работу командного файла

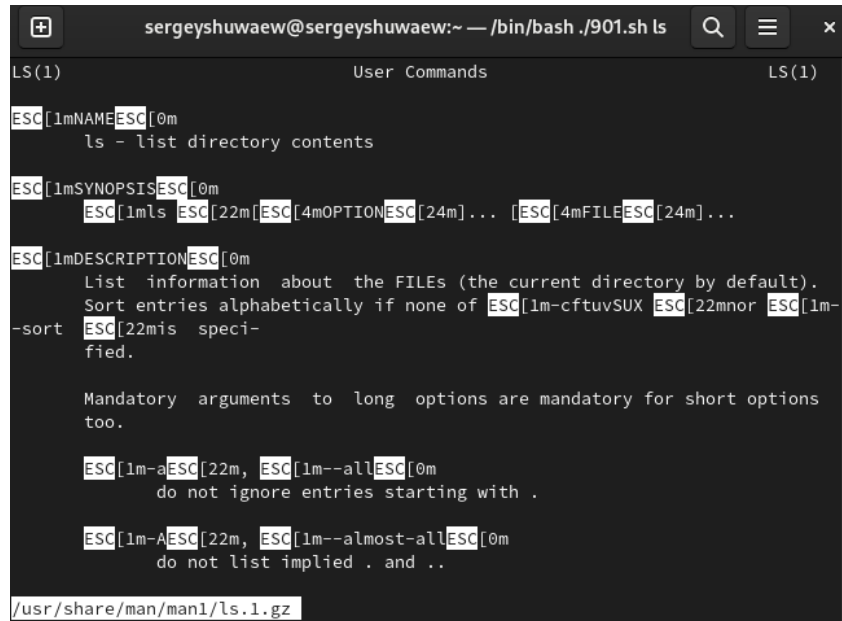
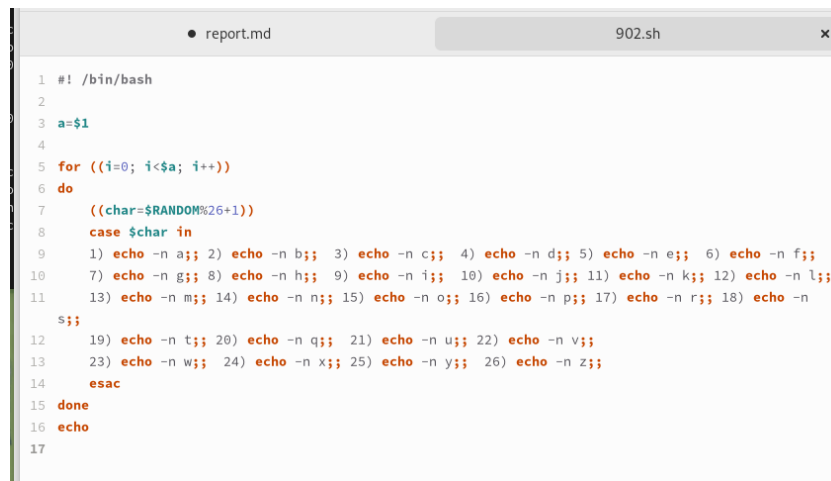


Рис. 4.6: Командный файл работает так же, как и команда man, открывает справку по указанной утилите

```
[sergeyshuwaew@sergeyshuwaew ~]$ touch 902.sh
[sergeyshuwaew@sergeyshuwaew ~]$ chmod +x 902.sh
[sergeyshuwaew@sergeyshuwaew ~]$ bash 902.sh
902.sh: строка 5: ((: i<: синтаксическая ошибка: ожидается операнд (неверный маркер «<»)
[sergeyshuwaew@sergeyshuwaew ~]$ bash 902.sh 20 ghfkhgkdngkdfkg
pkjvxafulybyssdgdti
[sergeyshuwaew@sergeyshuwaew ~]$
```

Рис. 4.7: Создаю файл для кода третьей программы, пишу программу и проверяю ее работу

A screenshot of a terminal window with two tabs: 'report.md' and '902.sh'. The '902.sh' tab is active, displaying a shell script. The script starts with a shebang line, sets a variable 'a' to 1, and enters a 'for' loop that iterates from 0 to 'a'. Inside the loop, a random character is generated using '\$RANDOM%26+1' and assigned to 'char'. A 'case' statement then maps the numeric value of 'char' to a specific letter of the alphabet (a-z). Each letter is printed on a new line using 'echo -n'. The script ends with an 'echo' command on the final line.

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

Рис. 4.8: Используя встроенную переменную \$RANDOM, пишу командный файл, генерирующий случайную последовательность букв латинского алфавита. Т.к. \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767, ввожу ограничения так, чтобы была генерация чисел от 1 до 26

## 5 Выводы

При выполнении данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX, научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 6 Ответы на 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++)) 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 и недостатки 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