

Лабораторная работа №12

Программирование в командном процессоре ОС UNIX. Расширенное
программирование

Кузнецова София Вадимовна

Содержание

Цель работы	5
Задание	6
Выполнение лабораторной работы	7
Контрольные вопросы	14
Выводы	17

Список иллюстраций

0.1	Создание первого файла для скрипта	7
0.2	Написание первого скрипта	8
0.3	Право на выполнение, запуск файла и проверка	8
0.4	Доработка первого скрипта	9
0.5	Сохранение и проверка	10
0.6	Создание файла для второго скрипта	10
0.7	Написание второго скрипта	11
0.8	Право на выполнение, запуск файла для команды ls	11
0.9	Информация о команде ls	12
0.10	Создание третьего файла	12
0.11	Написание третьего скрипта	12
0.12	Право на выполнение, запуск файла	13

Список таблиц

Цель работы

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

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 имела 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.

Выполнение лабораторной работы

Откроем терминал и создадим в домашнем каталоге файл `sem.sh`. После чего перейдём в `emacs`.

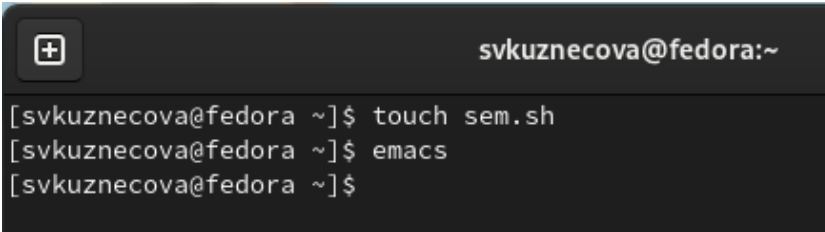
A screenshot of a terminal window with a dark background. The title bar at the top shows a window icon on the left and the text 'svkuznecova@fedora:~' on the right. The terminal content shows three lines of text: the first line is '[svkuznecova@fedora ~]\$ touch sem.sh', the second line is '[svkuznecova@fedora ~]\$ emacs', and the third line is '[svkuznecova@fedora ~]\$'.

Рис. 0.1: Создание первого файла для скрипта

В `emacs` откроем созданный файл `sem.sh` и приступим к написанию командного файла, который реализует 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.

```
sem.sh - GNU Emacs at fedora
File Edit Options Buffers Tools Sh-Script Help
[Icons: Open, Save, Undo, Cut, Copy, Paste, Find]

#!/bin/bash
t1=$1
t2=$2
s1=$(date +%s)
s2=$(date +%s)
((t=$s2-$s1))
while ((t < t1))
do
    echo "Ожидание"
    sleep 1
    s2=$(date +%s)
    ((t=$s2-$s1))
done
s1=$(date +%s)
s2=$(date +%s)
while ((t < t2))
do
    echo "Выполнение"
    sleep 1
    s2=$(date +%s)
    ((t=$s2-$s1))
done
_
```

Рис. 0.2: Написание первого скрипта

После того как скрипт написан мы сохраняем файл и запускаем emacs. В терминале мы даём этому файлу право на выполнение и запускаем его.

```
[svkuznecova@fedora ~]$ chmod +x sem.sh
[svkuznecova@fedora ~]$ ./sem.sh 4 7
Ожидание
Ожидание
Ожидание
Ожидание
```

Рис. 0.3: Право на выполнение, запуск файла и проверка

Теперь нам нужно доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов. Снова открываем emacs и наш файл sem.sh.



```
sem.sh - GNU Emacs at fedora
File Edit Options Buffers Tools Sh-Script Help
[Icons: Open, Save, Undo, Cut, Copy, Paste, Find]

function ogidania
{
  t1=$1
  t2=$2
  s1=$(date +%s)
  s2=$(date +%s)
  ((t=s2-s1))
  while ((t < t1))
  do
    echo "Ожидание"
    sleep 1
    s2=$(date +%s)
    ((t=s2-s1))
  done
}

function vipolnenie
{
  s1=$(date +%s)
  s2=$(date +%s)
  while ((t < t2))
  do
    echo "Выполнение"
    sleep 1
    s2=$(date +%s)
    ((t=s2-s1))
  done
}

t1=$1
t2=$2
command=$3
while true
do
  if [ "$command" == "Выход" ]
  then
    echo "Выход"
    exit 0
  fi
  if [ "$command" == "Ожидание" ]
  then ogidania
  fi
  if [ "$command" == "Выполнение" ]
  then vipolnenie
  fi
  echo "Следующее действие: "
  read command
done
```

Рис. 0.4: Доработка первого скрипта

Сохраняем файл и проверяем его работу.

```

ismahorin@fedora ~]$ emacs
ismahorin@fedora ~]$ chmod +x sem.sh
ismahorin@fedora ~]$ ./sem.sh 2 3 Ожидание > /dev/pts/1 &
[1] 132301
bash: /dev/pts/1: Отказано в доступе
[1]+ Выход 1          ./sem.sh 2 3 Ожидание > /dev/pts/1
ismahorin@fedora ~]$ ./sem.sh 2 3 Ожидание > /dev/pts/2 &
[1] 132313
bash: /dev/pts/2: Отказано в доступе
[1]+ Выход 1          ./sem.sh 2 3 Ожидание > /dev/pts/2
ismahorin@fedora ~]$ bash: /dev/pts/2: Отказано в доступе
bash: bash:: command not found...
similar command is: 'bash'
ismahorin@fedora ~]$

```

Рис. 0.5: Сохранение и проверка

В домашнем каталоге создаём файл `man.sh`, но уже для второго скрипта. Запускаем `emacs`.

```

+ svkuznecova@fedora:~
[svkuznecova@fedora ~]$ touch man.sh
[svkuznecova@fedora ~]$ emacs
[svkuznecova@fedora ~]$

```

Рис. 0.6: Создание файла для второго скрипта

Приступаем к реализации команды `man` с помощью командного файла. Изучим 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`.

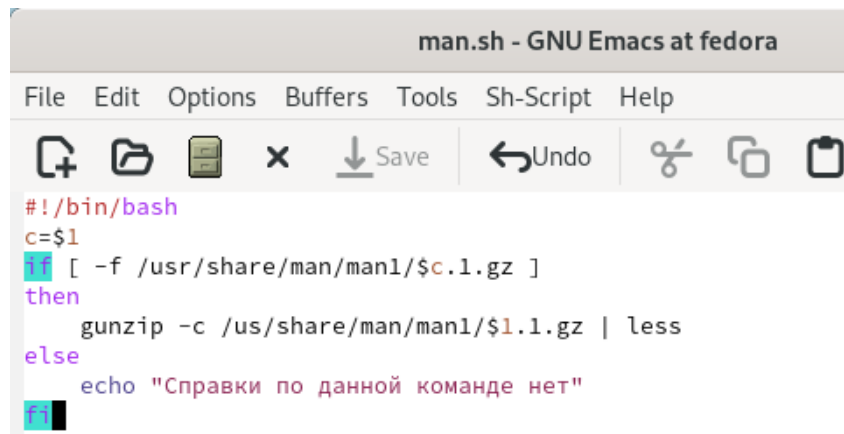


Рис. 0.7: Написание второго скрипта

Сохраняем файл и даём в терминале право на выполнение. Запускаем файл `man.sh` для команды `ls`.

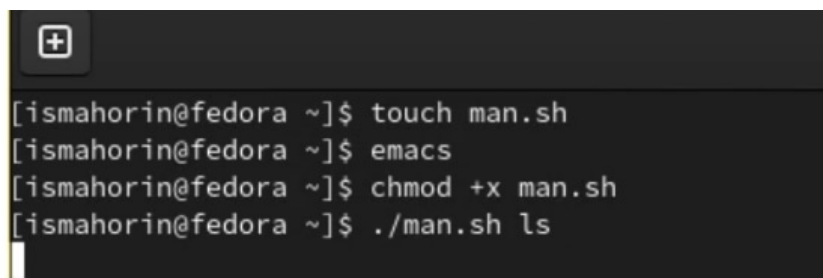


Рис. 0.8: Право на выполнение, запуск файла для команды `ls`

```

+
\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
TH LS "1" "March 2022" "GNU coreutils 8.32" "User Commands"
SH NAME
ls \- list directory contents
SH SYNOPSIS
B ls
\fI\,OPTION\|fR\... [\fI\,FILE\|fR\...
SH DESCRIPTION
\" Add any additional description here
PP
list information about the FILEs (the current directory by default).
sort entries alphabetically if none of \fB\--cftuvSUX\|fR nor \fB\--sort\|fR is specified.
PP
andatory arguments to long options are mandatory for short options too.
TP
\fB\-a\|fR, \fB\--all\|fR
do not ignore entries starting with .
TP
\fB\-A\|fR, \fB\--almost-all\|fR
do not list implied . and ..
TP
\fB\--author\|fR
with \fB\-l\|fR, print the author of each file
TP

```

Рис. 0.9: Информация о команде ls

Снова в домашнем каталоге создаём файл. Запускаем emacs.

```

+ svkuznecova@fedora:~ — emacs
[svkuznecova@fedora ~]$ touch random.sh
[svkuznecova@fedora ~]$ emacs

```

Рис. 0.10: Создание третьего файла

После открытия файла random.sh напишем командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтём, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

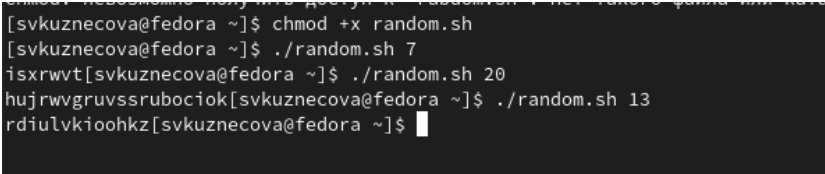
```

random.sh - GNU Emacs at fedora
File Edit Options Buffers Tools Sh-Script Help
[Icons] [Save] [Undo] [Find]
#!/bin/bash
set -f
for (( i=0; i<5; i++ ))
do
  (( chars=$RANDOM%26+1 ))
  case $chars in
    1) echo -n a;; 2) echo -n b;; 3) echo -n c;; 4) echo -n d;; 5) echo -n e;; 6) echo -n f;; 7) echo -n g;; 8) echo -n h;; 9) echo -n i;;
    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;; 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;; 26) echo -n z;;
  esac
done
echo

```

Рис. 0.11: Написание третьего скрипта

Сохраняем наш скрипт и даём право на выполнение. Запускаем файл для трёх разных чисел.

A terminal window with a dark background and light-colored text. It shows a series of commands and their outputs. The first command is 'chmod +x random.sh', followed by three invocations of './random.sh' with arguments 7, 20, and 13. Each invocation produces a long, seemingly random alphanumeric string as output.

```
[svkuznecova@fedora ~]$ chmod +x random.sh
[svkuznecova@fedora ~]$ ./random.sh 7
isxrwvt[svkuznecova@fedora ~]$ ./random.sh 20
hujrwwgruvssrubociok[svkuznecova@fedora ~]$ ./random.sh 13
rdiulvkiioohkz[svkuznecova@fedora ~]$
```

Рис. 0.12: Право на выполнение, запуск файла

Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке:

```
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" Результат: Hello, World
```

- Второй:

```
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` 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.

Выводы

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