

## Лабораторная работа № 14

Программирование в командном процессоре ОС UNIX. Расширенное программирование

---

Баптишта Матеуж Андре ; Нкабв-01-23

Российский университет дружбы народов, Москва, Россия

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

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.

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:

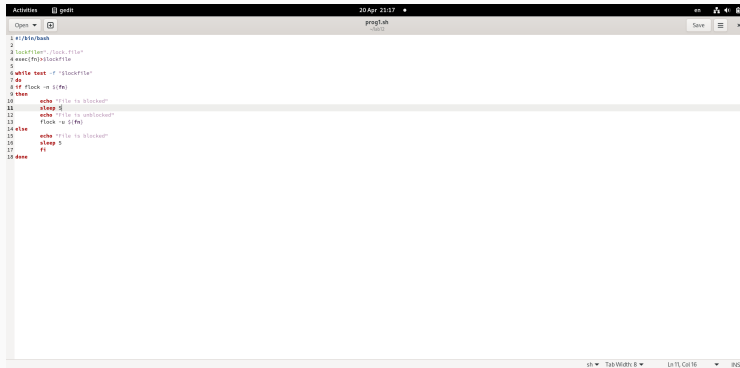
- оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
- C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (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

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?**).

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. (рис. (fig:001?; fig:002?))

# Выполнение лабораторной работы



The screenshot shows a gedit text editor window titled 'prog1.sh' with a 'Save' button and a close icon. The editor contains a shell script with the following content:

```
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 unblocked"
13     flock -u $(&fn)
14   else
15     echo "file is blocked"
16     sleep 5
17   fi
18 done
```

The status bar at the bottom indicates the shell is 'sh', tab width is 8, and the cursor is at line 16, column 16.

Рис. 1: Текст первой программы

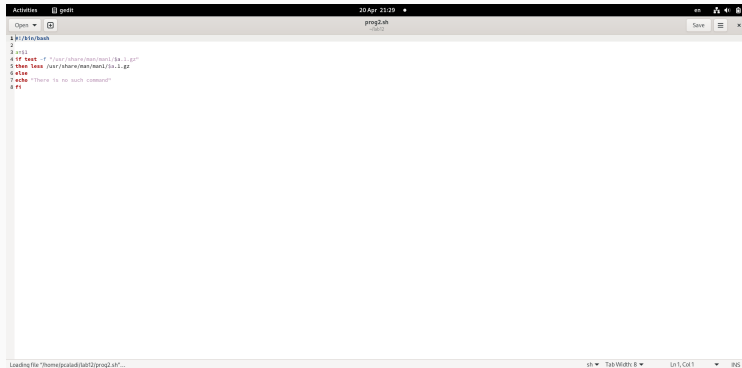


## Выполнение лабораторной работы

A screenshot of a terminal window titled "Terminal" with a timestamp of "20 Apr 21:17". The terminal shows a user running a command: `|pcalodofedera lab1215 bash prog1.sh`. The output is a continuous loop of error messages: `prog1.sh line 4: execve(): command not found`, `clock: requires file descriptor, file or directory`, and `File is blocked`. This sequence of errors repeats multiple times, indicating a deadlock or a loop where the program keeps attempting to execute a command that fails due to resource constraints or a system error.

Рис. 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. (fig:003?; fig:004?; fig:005?))



```
1 #!/bin/bash
2
3 setf
4 if test -f "/usr/share/man/man1/ls.1.gz"
5 then Test "/usr/share/man/man1/ls.1.gz"
6 else
7 echo "There is no such command"
8 fi
```

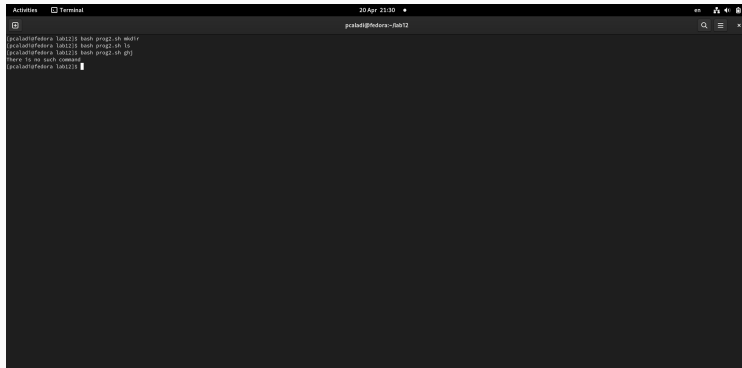
Рис. 3: Текст второй программы

## Выполнение лабораторной работы

[illegible]

Рис. 4: Результат

# Выполнение лабораторной работы

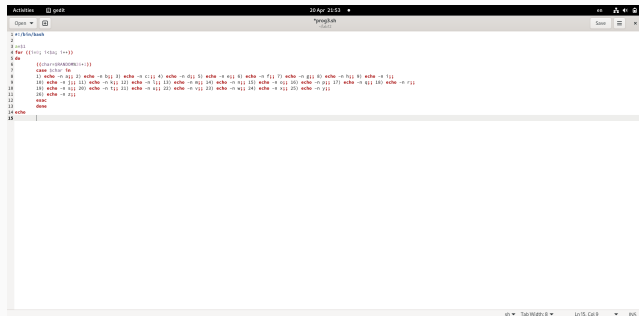


A terminal window titled "Terminal" with a timestamp of "20 Apr 21:30". The window shows the following commands and their outputs:

```
pcalad@fedora-lab12:~$ bash prog2.sh mkdir
pcalad@fedora-lab12:~$ bash prog2.sh ls
pcalad@fedora-lab12:~$ bash prog2.sh dj
there is no such command
pcalad@fedora-lab12:~$
```

Рис. 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. (fig:006?; fig:007?))



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

Рис. 6: Текст третьей программы

## Выполнение лабораторной работы

[illegible]

Рис. 7: Результат

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



Спасибо за внимание!