

Лабораторная работа № 12

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Ариоке Габриэль Одафе

20 Апрель 2023

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

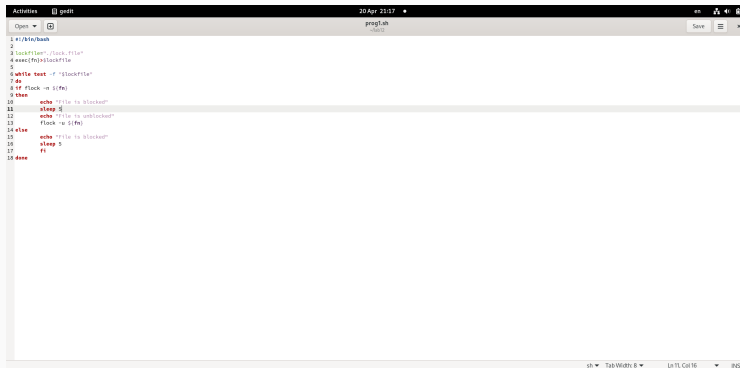
POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных 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 имелась возможность взаимодействия трёх и более процессов.

(рис. 1, 2)

Выполнение лабораторной работы



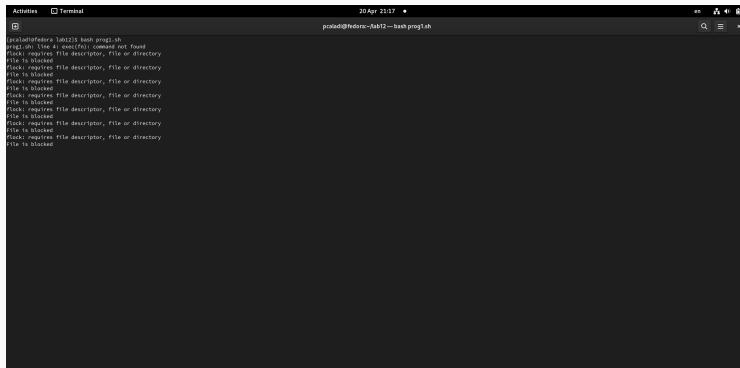
The screenshot shows a terminal window titled 'Activities' and 'edit'. The window contains a shell script for file locking. The script is as follows:

```
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
```

The terminal window also shows the date and time '20 Apr 21:57' and the user 'prog1.sh'.

Figure 1: Текст первой программы

Выполнение лабораторной работы

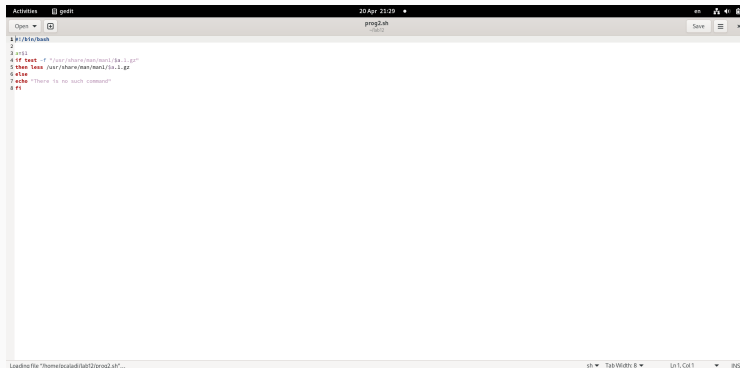


```
pcalad@fedora lab12:~$ bash prog1.sh
prog1.sh: line 4: exec(fn): command not found
flock: requires file descriptor, file or directory
File is blocked
flock: requires file descriptor, file or directory
File is blocked
flock: requires file descriptor, file or directory
File is blocked
flock: requires file descriptor, file or directory
File is blocked
flock: requires file descriptor, file or directory
File is blocked
flock: requires file descriptor, file or directory
File is blocked
flock: requires file descriptor, file or directory
File is blocked
flock: requires file descriptor, file or directory
File is blocked
```

Figure 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. 3, 4, 5)

Выполнение лабораторной работы



```
Activities | edit 20 Apr 21:29 en
prog2.sh
Save
1 #!/bin/bash
2
3 #11
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
```

Loading file "/home/pclad/lab12/prog2.sh"...

sh Tab-Width: 8 Ln 1, Col 1 INS

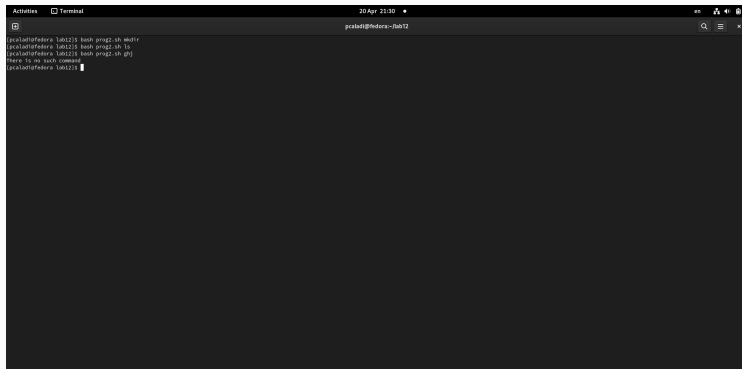
Figure 3: Текст второй программы

Выполнение лабораторной работы

[illegible]

Figure 4: Результат

Выполнение лабораторной работы

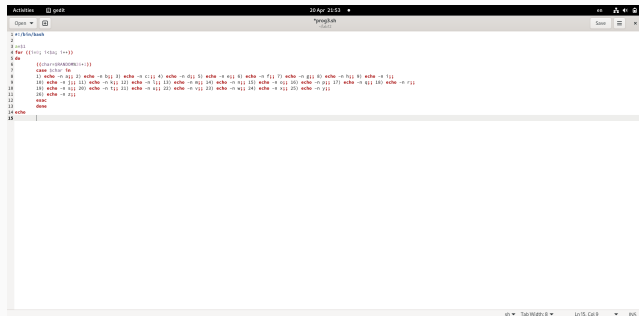


A terminal window titled "Terminal" with a timestamp of "20 Apr 21:30". The window shows the following commands and their outputs:

```
pcalad@fedora lab12$ bash prog2.sh mkdir
pcalad@fedora lab12$ bash prog2.sh ls
pcalad@fedora lab12$ bash prog2.sh git
there is no such command
pcalad@fedora lab12$
```

Figure 5: Результат

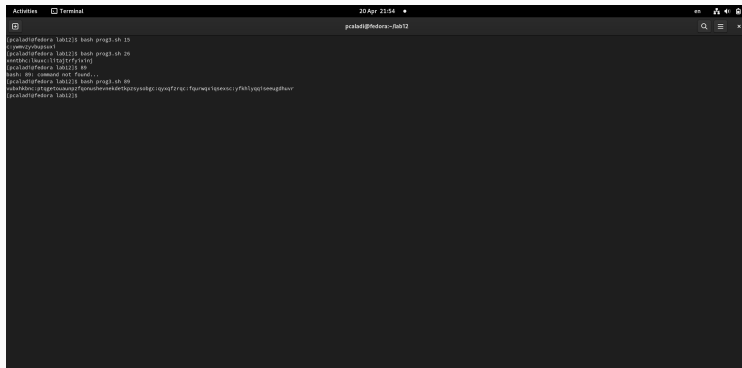
3. Используя встроенную переменную \$RANDOM, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767. (рис. 6, 7)



```
1 #!/bin/bash
2
3 seed=1
4 for ((i=0; i<100; i++))
5 do
6     ((seed=$RANDOM*33))
7     case $seed in
8         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;;
9         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;;
10        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;;
11        26) echo -n z;;
12    esac
13    done
14 echo
```

Figure 6: Текст третьей программы

Выполнение лабораторной работы



```
pcalad@fedora:~$ bash prog1.sh 15
c:\pawzy\pawseel
[pcalad@fedora ~]$ bash prog1.sh 20
montecarlo[110]rffnln
[pcalad@fedora ~]$ 89
bash: 89: command not found...
[pcalad@fedora ~]$ bash prog1.sh 89
subakhdcc:ptagetouanzf6onushnvekkethkzysyabgr:qyqfzrqc:fQurwjs1exexic:yfkhlyqptomagdhuvr
[pcalad@fedora ~]$
```

Figure 7: Результат

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

Спасибо за внимание!