

Лабораторная работа-12

Программирование в командном процессоре ОС UNIX. Расширенное
программирование

Арсоева Залина НБИбд-01-21

Содержание

Цель работы	5
Задания	6
Ход работы:	7
Вывод:	13
Ответы на контрольные вопросы:	14

Список иллюстраций

0.1	Создаю файл	7
0.2	Пишу скрипт	8
0.3	Программа	8
0.4	Создаю файл	9
0.5	Написала скрипт	9
0.6	Программа	10
0.7	Программа	10
0.8	Программа	11
0.9	Создаю текстовый файл	11
0.10	Пишу скрипт	12
0.11	Программа	12

Список таблиц

Цель работы

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

Задания

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.

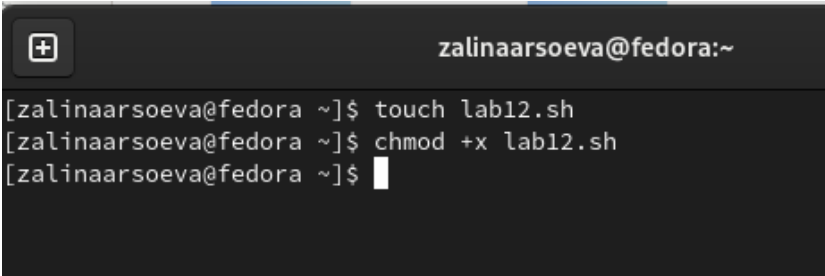
2. Реализовать команду `man` с помощью командного файла. Изучите содержимое каталога `/usr/share/man/man1`. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой `less` сразу же просмотрев содержимое

файл должен получать в виде аргумента командной строки название команды 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.

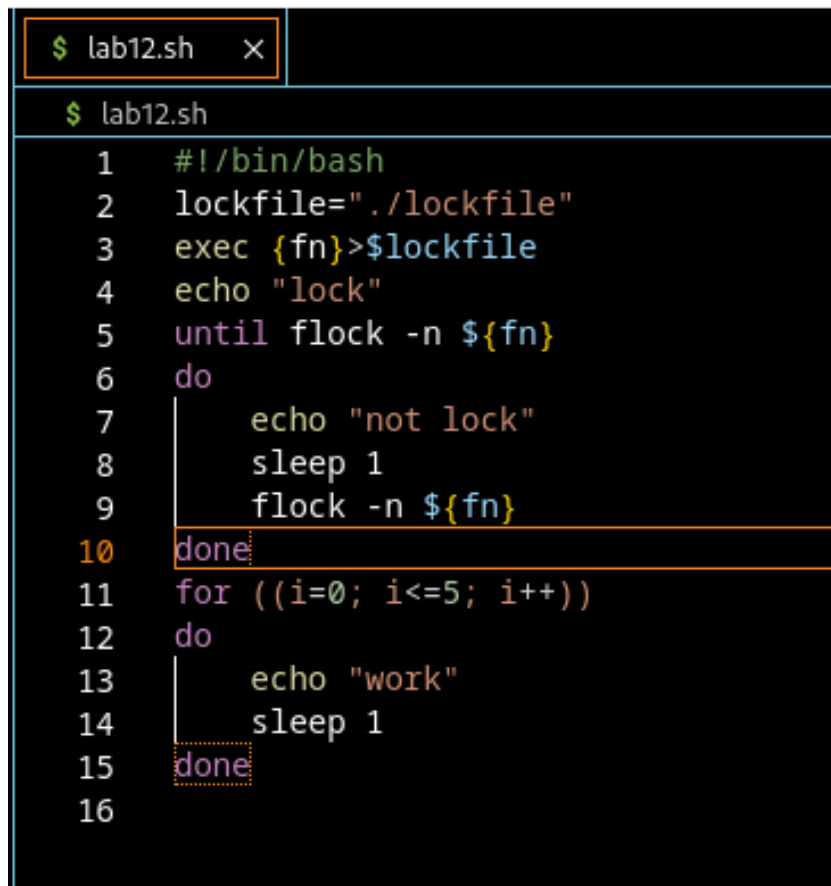
Ход работы:

1. Написала 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). Запустил 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])(рис. [-@fig:002])(рис. [-@fig:003])



```
zalinaarsoeva@fedora:~  
[zalinaarsoeva@fedora ~]$ touch lab12.sh  
[zalinaarsoeva@fedora ~]$ chmod +x lab12.sh  
[zalinaarsoeva@fedora ~]$
```

Рис. 0.1: Создаю файл



```
$ lab12.sh ×
$ lab12.sh
1  #!/bin/bash
2  lockfile="./lockfile"
3  exec {fn}>$lockfile
4  echo "lock"
5  until flock -n ${fn}
6  do
7      echo "not lock"
8      sleep 1
9      flock -n ${fn}
10 done
11 for ((i=0; i<=5; i++))
12 do
13     echo "work"
14     sleep 1
15 done
16
```

Рис. 0.2: Пишу скрипт




```
zalinaarsoeva@fedora:~ — /bin/bash ./lab12.sh
[zalinaarsoeva@fedora ~]$ touch lab12.sh
[zalinaarsoeva@fedora ~]$ chmod +x lab12.sh
[zalinaarsoeva@fedora ~]$ ./lab12.sh
lock
work
work
work
work
work
work

```

Рис. 0.3: Программа

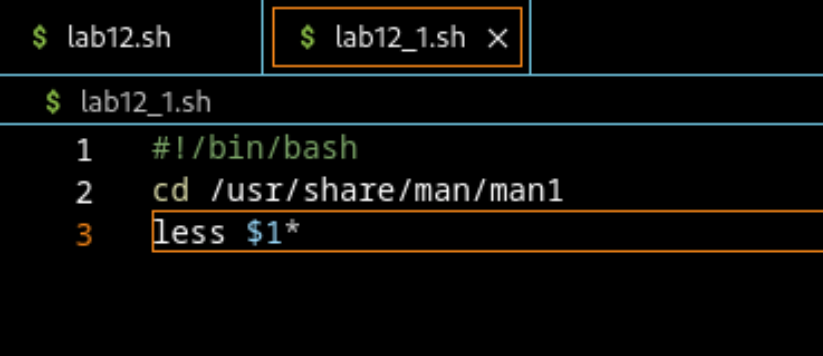
2. Реализовала команду `map` с помощью командного файла. Изучил содержимое

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:004])(рис. [-@fig:005])(рис. [-@fig:006])(рис. [-@fig:007])(рис. [-@fig:008])



```
zalinaarsoeva@fedora:~  
[zalinaarsoeva@fedora ~]$ touch lab12.sh  
[zalinaarsoeva@fedora ~]$ chmod +x lab12.sh  
[zalinaarsoeva@fedora ~]$ ./lab12.sh  
lock  
work  
work  
work  
work  
work  
work  
work  
[zalinaarsoeva@fedora ~]$
```

Рис. 0.4: Создаю файл



```
$ lab12.sh $ lab12_1.sh X  
$ lab12_1.sh  
1  #!/bin/bash  
2  cd /usr/share/man/man1  
3  less $1*
```

Рис. 0.5: Написала скрипт

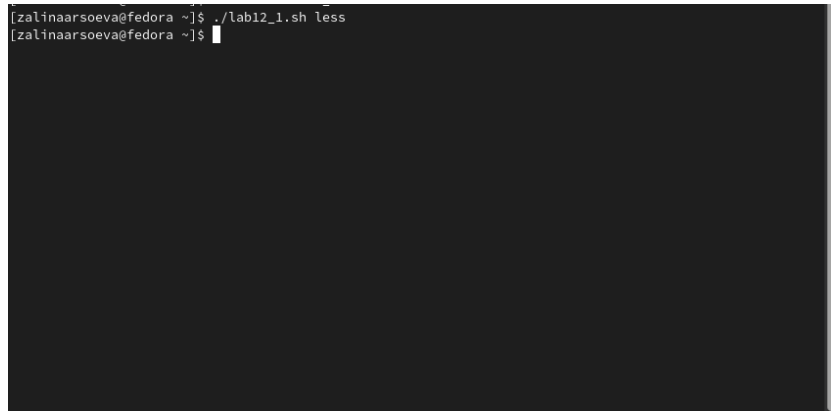


Рис. 0.6: Программа



Рис. 0.7: Программа

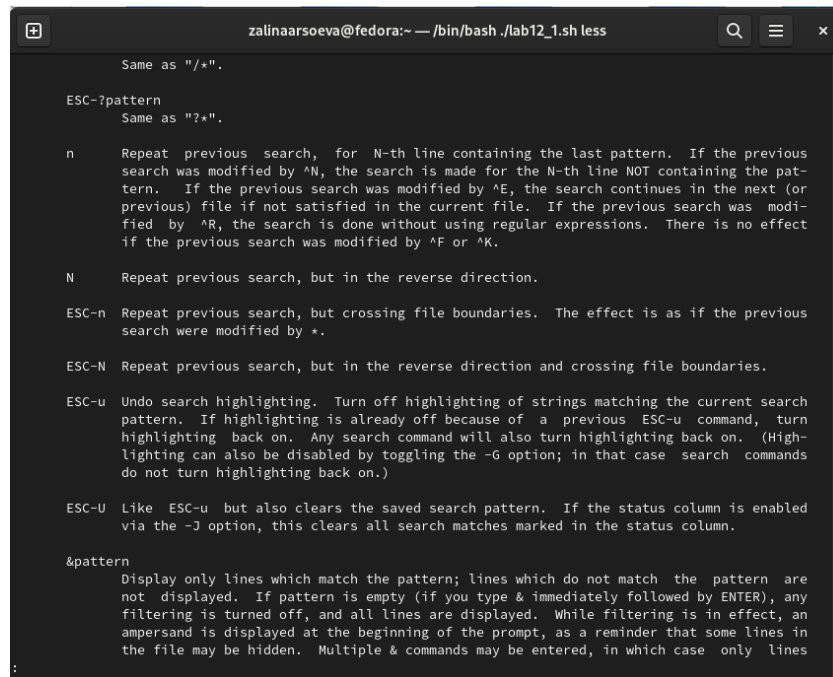


Рис. 0.8: Программа

3. Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. Учла, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.(рис. [-@fig:009])(рис. [-@fig:010])(рис. [-@fig:011])

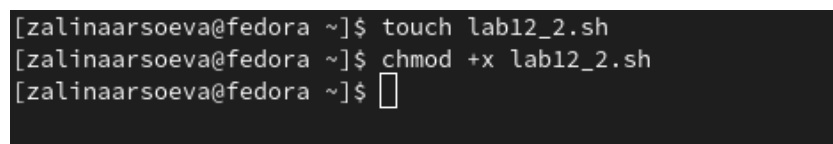


Рис. 0.9: Создаю текстовый файл

```
$ lab12.sh $ lab12_1.sh $ lab12_2.sh x
$ lab12_2.sh
1  #!/bin/bash
2  M=10
3  c=1
4  d=1
5  echo
6  echo "10 random worlds: "
7  while (($c!=($M+1)))
8  do
9      echo $(for((i=1; i<=10; i++)); do printf '%s' "${RANDOM:0:1}"; done) | tr '[0-9]' '[a-z]'
10     echo $d
11     ((c+=1))
12     ((d+=1))
13 done
```

Рис. 0.10: Пишу скрипт

```
[zalinaarsoeva@fedora ~]$ ./lab12_2.sh
10 random worlds:
cbccbcfdcb
1
cdcbbbbfgc
2
bfdcbicbdb
3
dcdbeccdj
4
dbdbjccgdb
5
bjdfhccchc
6
cbbbcdbjc
7
cccgbcbgcc
8
iiccbbedcb
9
cbdbcbibbg
10
[zalinaarsoeva@fedora ~]$
```

Рис. 0.11: Программа

Вывод:

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

Ответы на контрольные вопросы:

1. В строке `while [$1 != "exit"]` квадратные скобки надо заменить на круглые.
2. Есть несколько видов конкатенации строк. Например, `VAR1="Hello," VAR2="World" VAR3="$VAR1$VAR2" echo "$VAR3"`
3. Команда `seq` выводит последовательность целых или действительных чисел, подходящую для передачи в другие программы. В `bash` можно использовать `seq` с циклом `for`, используя подстановку команд. Например, `$ for i in $(seq 1 0.5 4) do echo "The number is $i" done`
4. Результатом вычисления выражения `$(10/3)` будет число 3.
5. Список того, что можно получить, используя `Z Shell` вместо `Bash`: Встроенная команда `zmv` поможет массово переименовать файлы/директории, например, чтобы добавить `‘.txt’` к имени каждого файла, запустите `zmv -C '(*)(#q.)' '$1.txt'`. Утилита `zcalc` — это замечательный калькулятор командной строки, удобный способ считать быстро, не покидая терминал. Команда `zparseopts` — это однострочник, который поможет разобрать сложные варианты, которые предоставляются скрипту. Команда `autopushd` позволяет делать `popd` после того, как с помощью `cd`, чтобы вернуться в предыдущую директорию. Поддержка чисел с плавающей точкой (к которой `Bash` не содержит). Поддержка для структур данных «хэш». Есть также ряд особенностей, которые присутствуют только в `Bash`: Опция командной строки `-porc`, которая позволяет пользователю иметь дело с инициализацией командной строки, не читая файл `.bashrc`. Использование опции `-rcfile` с `bash` позволяет исполнять

команды из определённого файла. Отличные возможности вызова (набор опций для командной строки) Может быть вызвана командой `sh` Bash можно запустить в определённом режиме POSIX. Примените `set -o posix`, чтобы включить режим, или `—posix` при запуске. Можно управлять видом командной строки в Bash. Настройка переменной `PROMPT_COMMAND` с одним или более специальными символами настроит её за вас. Bash также можно включить в режиме ограниченной оболочки (с `rbash` или `—restricted`), это означает, что некоторые команды/действия больше не будут доступны: Настройка и удаление значений служебных переменных `SHELL`, `PATH`, `ENV`, `BASH_ENV` Перенаправление вывода с использованием операторов `'>'`, `'>|'`, `'<>'`, `'>&'`, `'&>'`, `'>'` Разбор значений `SHELLOPTS` из окружения оболочки при запуске Использование встроенного оператора `exes`, чтобы заменить оболочку другой командой

6. Синтаксис конструкции `for ((a=1; a <= LIMIT; a++))` верен.
7. Язык `bash` и другие языки программирования: -Скорость работы программ на ассемблере может быть более 50% медленнее, чем программ на `си/си++`, скомпилированных с максимальной оптимизацией; -Скорость работы виртуальной ява-машины с байт-кодом часто превосходит скорость аппаратуры с кодами, получаемыми трансляторами с языков высокого уровня. Ява-машина уступает по скорости только ассемблеру и лучшим оптимизирующим трансляторам; -Скорость компиляции и исполнения программ на яваскрипт в популярных браузерах лишь в 2-3 раза уступает лучшим трансляторам и превосходит даже некоторые качественные компиляторы, безусловно намного (более чем в 10 раз) обгоняя большинство трансляторов других языков сценариев и подобных им по скорости исполнения программ; -Скорость кодов, генерируемых компилятором языка `си` фирмы Intel, оказалась заметно меньшей, чем компилятора GNU и иногда LLVM; -Скорость ассемблерных кодов `x86-64` может меньше, чем аналогичных кодов `x86`, примерно на 10%;

-Оптимизация кодов лучше работает на процессоре Intel; -Скорость исполнения на процессоре Intel была почти всегда выше, за исключением языков лисп, эрланг, аук (gawk, mawk) и бэш. Разница в скорости по бэш скорее всего вызвана разными настройками окружения на тестируемых системах, а не собственно транслятором или железом. Преимущество Intel особенно заметно на 32-разрядных кодах; -Стек большинства тестируемых языков, в частности, ява и яваскрипт, поддерживают только очень ограниченное число рекурсивных вызовов. Некоторые трансляторы (gcc, iss, ...) позволяют увеличить размер стека изменением переменных среды исполнения или параметром; -В рассматриваемых версиях gawk, php, perl, bash реализован динамический стек, позволяющий использовать всю память компьютера. Но perl и, особенно, bash используют стек настолько экстенсивно, что 8-16 ГБ не хватает для расчета ask(5,2,3)