

Отчёт по лабораторной работе 12

Программирование в командной процессоре ОС Linux. Расширенное
программирование.

Фомичева Маргарита Романовна

Содержание

1	Цель работы	5
2	Ход работы	6
3	Вывод	12
4	Ответы на контрольные вопросы	13

Список таблиц

Список иллюстраций

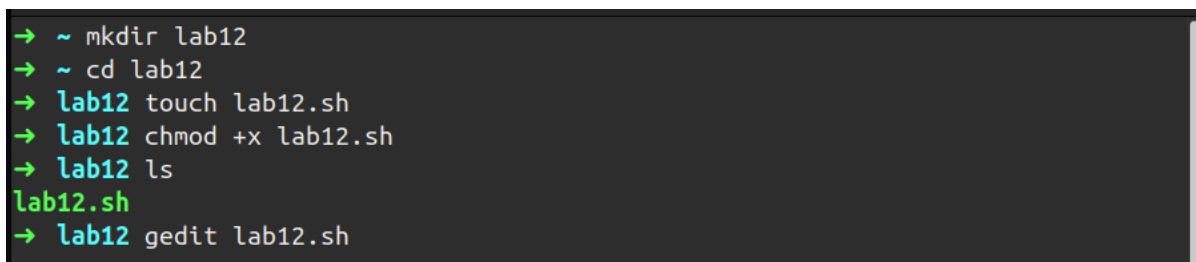
2.1	1	6
2.2	2	7
2.3	3	7
2.4	4	8
2.5	5	8
2.6	6	8
2.7	7	9
2.8	8	10
2.9	9	10
2.10	10	11

1 Цель работы

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

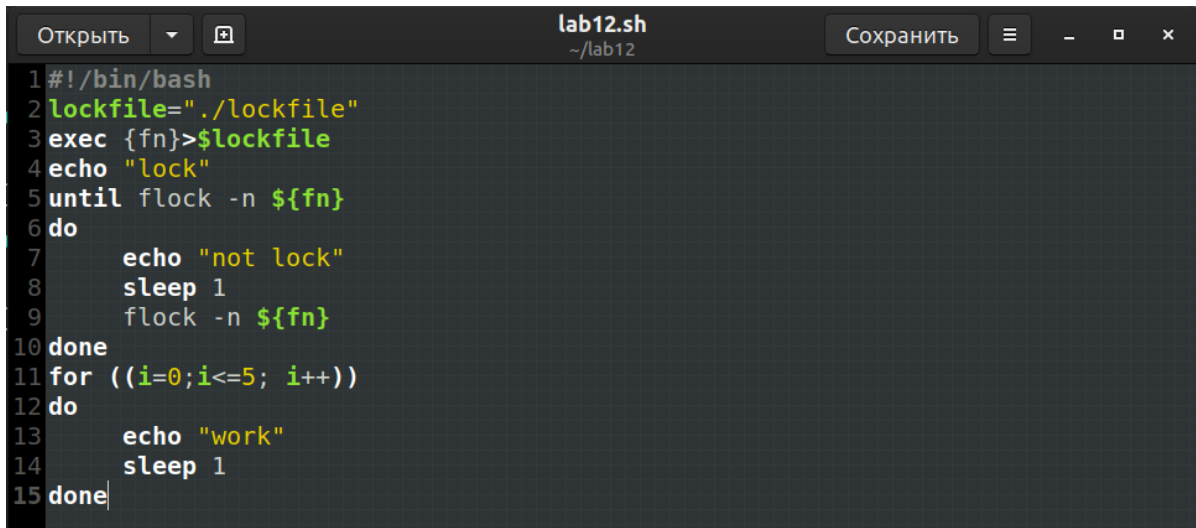
2 Ход работы

- 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 $t_2 < t_1$, а также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустила 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 в фоновом, а в привилегированном режиме. Доработала программу так, чтобы имела возможность взаимодействия трёх и более процессов.



```
→ ~ mkdir lab12
→ ~ cd lab12
→ lab12 touch lab12.sh
→ lab12 chmod +x lab12.sh
→ lab12 ls
lab12.sh
→ lab12 gedit lab12.sh
```

Рис. 2.1: 1



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

Рис. 2.2: 2



```
→ lab12 ./lab12.sh
lock
work
work
work
work
work
work
work
→ lab12
```

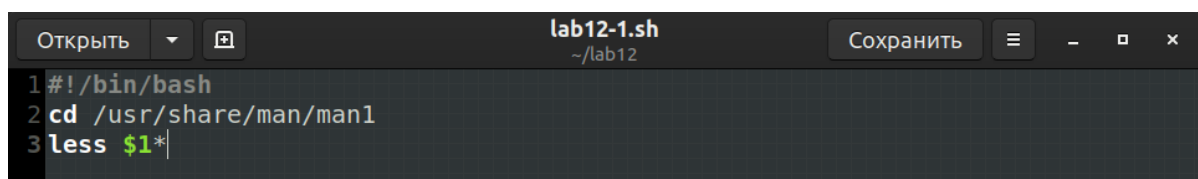
Рис. 2.3: 3

- Реализовала команду man с помощью командного файла. Изучила содержимое каталога /usr/share/man/man1. В нём 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.

```
→ lab12 touch lab12-1.sh
→ lab12 chmod +x lab12-1.sh
→ lab12 ls
1.png 2.png 3.png lab12-1.sh lab12.sh lockfile
→ lab12
```

Рис. 2.4: 4



```
Открыть  lab12-1.sh  Сохранить
~/lab12
1#!/bin/bash
2cd /usr/share/man/man1
3less $1*
```

Рис. 2.5: 5

```
→ ~ cd lab12
→ lab12 ./lab12-1.sh less
```

Рис. 2.6: 6


```
./lab12-1.sh less
^Tj^D<FC><D1>#<94>1k<DE><D6>8^_Y^U<F3><FE><B8><8B><8B>r^U<C5><^<A7>*<9E>*<v^X
G<87>|<92>wU<92><DA>y<A6><91>T<B6><86><BB><95>,4Sc<9B><CB>♦♦♦♦Z^V5b11^U<E8>q^BK<
<C0><86>$<A5><99>^Y<A2>Z5^E<BA>^U<B9><91>^C^G<A7>[<^0^0<F0><D0><D7>/<F0><DF>1<FD>
<F9>[<8E>^F#<B8>|<86>P^W ]<C0>^S<92><A8>.Pk<AC><EB><AC> yR,T<C7>^Zo5.z<EF><99>S
;<84><D8>12<89>t<9B>[<91>4<9B><BE>7] <80><92>-<99>^](<B6><F2>n\g<B8>M.<8F>^<FC>I
<97>9<B0>j<B7><A5>]<C8>|<C4>Z<A3><BD><B1>s<90>%<80><90><81>,;K<DF>kBŮ^n<FB>><8E>
<9E><82>V+<BA><B5>X!<C1><92> <8C>S<93>tQc<87>^FHg<88><87>*7^KwU<C4>Z@!<A7>
<CC>:HK<B4>C<B4><E0>*<C1><AD><BA><AC><DC>p" <9C>^@<96>L^D<9C><8C>L<9E>
<F4><8A>q<8F>L<98><C8>=^X<CF>^Q^X^Zw<D0><F1><F9><C1>^Y<B2><E5><EC><E0><FC><F9>
<D9>11<88><FE>5<F4>'1<EF>^M<FD><F9><C3><F5>L<BB><C3>^Z$<D6>S^<93>6{r7X~<9C>{<82>
^BG<EF><DF>Lb<A6>:AB<B6><C9>aPG7<E2><82>^C^EgD<B6>^^<B2>:U
^LZ<9E><A8>:<D0> <C2>LH<98>=<C0>^?I-=<8A>C=<BA><FC>^SH5!<B9>^P<F1>!^L^?
<9F>D.<E9><BE>?*<A7><F8><EF>%<FD><F9><E4>^F<F7><FE><AC>b<F1>^Qm<FD><B9><AD><AD>o
<8A><A1><B3><B3>k<AA>Z^S<F2><CF>ol<FF><A8>|<FD> <{<C3><FC><C2>A4>y<8D><E0>
<D1>j8`Y<E4>.RI^f<A8><E4>^B<E3> ^<EF>g^Aty#@C3Is<A4><EC>^Fp(>;d$<8E>^^^
<9E>=<9D><FC><E4>)7-<CA><F4>=<80>^FIite:<99>V@^R0@A<82><8B>4^A<C7>N<8A><80>01^Le
<85>`5<D0><CE>^Q_c<E0><80><B8>^P1<C2>^K<B0><C4>x^A^R<9E>G<FC> ^A<A5>^?<9A><A2>E
<A4>0^E<83>2<81><AE>#^G^@<EE>^V<85><A9><82><E0>f2^U<FC>^F<82> <D4>^YM<8B>9<CB>Y
ZPd<95>+3^^C<C0>P<AC><83>^D<F3>go<A6>Wf<C6><FF>/<C8><E5>/<F5><81><F4><E6>fb<80>b
6^E#<B2>"<A6>^X<EE>BA>0<94>^L<CE><F8><95>0 <E5>Pk<F1>^X^V<D7>H<90>g<F5>.^W`
%#<CB><E0><E2><E8><CC><CC>5D<A8>^AQy<DD>7JJD^Ob<D3>^D^U^BS<88>a^M<FC>-^AY<8A>
<F4><FB><D1>sk<C8><E6>6<BA><82>t^GF<E6>^S <F9>JI<B2><86>^F<9F>\<A5>f<E1><ED><F2>
<D4>%[pi<CC>\U<E5>^Rf&#224;id<81>!^<F>{BBw^T{x^@x\<94>3" <C5>^<C2>3<B5>pQ9^E<B1>^DDp
:
```

Рис. 2.7: 7

- Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что \$RANDOM выдаёт вседслучайные числа в диапазоне от 0 до 32767.

```

→ lab12 touch lab12-2.sh
→ lab12 chmod +x lab12-2.sh
→ lab12 ls
1.png 3.png 5.png 7.png lab12-2.sh lockfile
2.png 4.png 6.png lab12-1.sh lab12.sh
→ lab12

```

Рис. 2.8: 8

```

Открыть  lab12-2.sh  Сохранить
~ /lab12
1#!/bin/bash
2M=10
3c=1
4d=1
5echo
6echo "10 random words:"
7while (($c!=($M+1)))
8do
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))
13done

```

Рис. 2.9: 9

```
→ lab12 ./lab12-2.sh
```

```
10 random words:
```

```
hiciccbdbc
```

```
1
```

```
gchbgchdbi
```

```
2
```

```
cbchbbbcdb
```

```
3
```

```
ccgdbbbjbb
```

```
4
```

```
bjchbbccgh
```

```
5
```

```
cjgbcbbbbc
```

```
6
```

```
bcccbefccb
```

```
7
```

```
cdbccbgcgb
```

```
8
```

```
bbbbebcbch
```

```
9
```

```
cjccihhddc
```

```
10
```

```
→ lab12
```

Рис. 2.10: 10

3 Вывод

- Я изучила основы программирования в оболочке ОС Unix. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

4 Ответы на контрольные вопросы

- В строке `while [$1 != "exit"]` квадратные скобки надо заменить на круглые.
- Есть несколько видов конкатенации строк. Например, `VAR1="Hello," VAR2="World" VAR3="$VAR1$VAR2" echo "$VAR3"`
- 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, `(seq 1 0.5 4) do echo "The number is $i" done`
- Результатом вычисления выражения `$((10/3))` будет число 3.
- 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 (коей `Bash` не содержит). Поддержка для структур данных «хэш». Есть также ряд особенностей, которые присутствуют только в `Bash`: Опция командной строки `-pois`, которая позволяет пользователю иметь дело с инициализацией командной строки, не читая файл `.bashrc`. Использование опции `-rfile` 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 (`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 `exec`, чтобы заменить оболочку другой командой

- Синтаксис конструкции `for ((a=1; a <= LIMIT; a++))` верен.
- 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)