Отчет по лабораторной работе №13

Тема:

Программирование в командном процессоре ОС UNIX. Расширенное программирование.

Российский Университет Дружбы Народов

Факультет Физико-Математических и Естественных Наук

Дисциплина: Операционные системы

Студент: Алших Маслем Ахмад

Группа: НФИБД-02-20

Москва, 2021г.

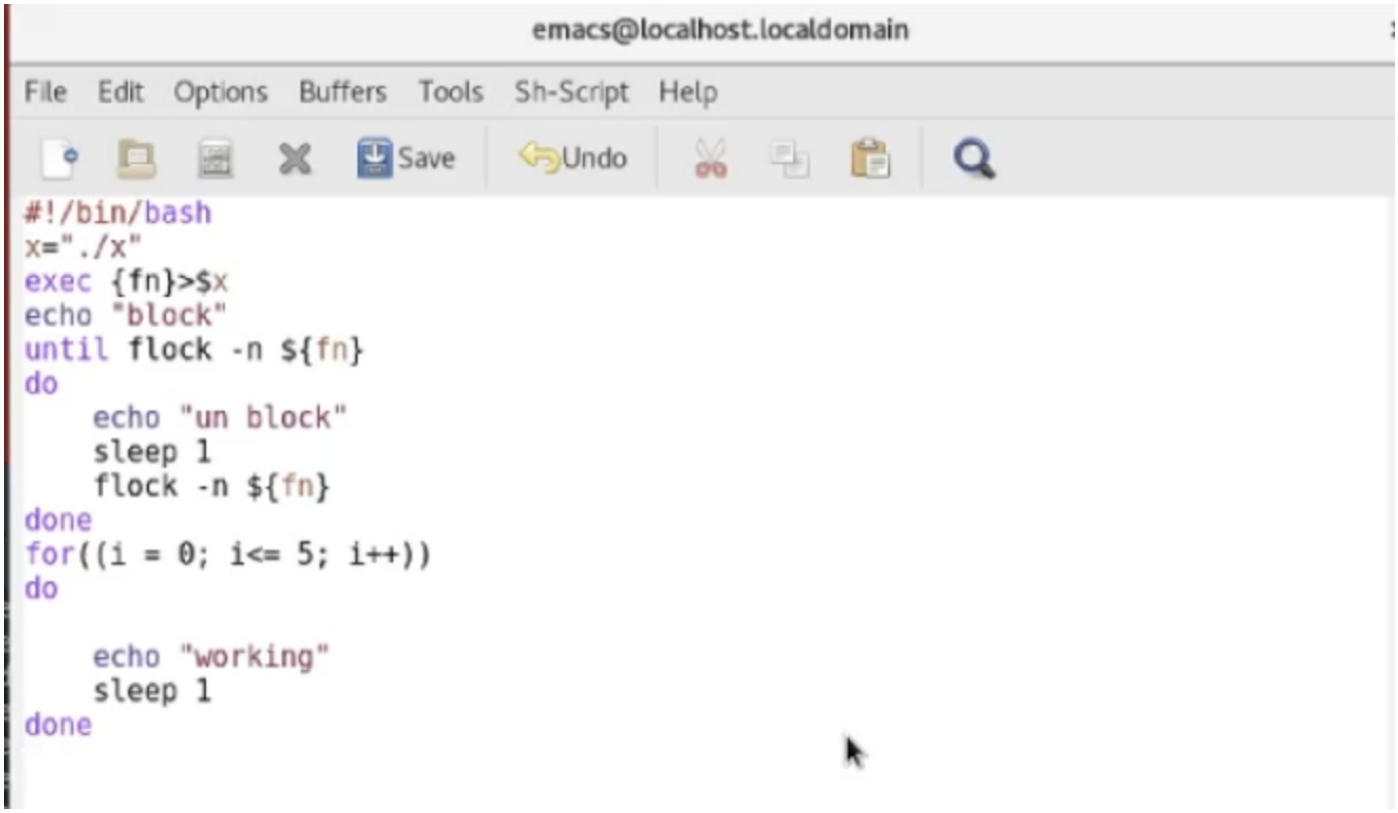
Цель работы

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

Ход работы:

1. Написал командный файл, реализующий упрощённый механизм семафоров. Командный файл ждет в течение некоторого времени *t1*, до освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовал его в течение некоторого времени *t2<>t1*, 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 в фоновом, а в привилегированном режиме. Доработал программу так, чтобы имелась возможность взаимодействия трёх и более процессов.

```
[ahmet@localhost ~]$ mkdir Lab13
[ahmet@localhost ~]$ cd Lab13
[ahmet@localhost Lab13]$ touch lab13.sh
[ahmet@localhost Lab13]$ chmod +x lab13.sh
[ahmet@localhost Lab13]$ emacs
```




```
[ahmet@localhost Lab13]$ bash lab13.sh
block
working
working
working
working
working
working
[ahmet@localhost Lab13]$
```

2. Реализовал команду *man* 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. Запустил командный файл. Командный файл получает в виде аргумента командной строки название команды и в виде результата выдавает справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1. Каждый архив можно открыть командой *less* сразу же просмотрев содержимое справки.

```
[ahmet@localhost Lab13]$ touch lab13_2.sh
[ahmet@localhost Lab13]$ chmod +x lab13_2.sh
[ahmet@localhost Lab13]$ emacs
```

emacs@localhost.localdomain

File Edit Options Buffers Tools Sh-Script Help

Save Undo

```
#!/bin/bash
cd /usr/share/man/man1
if (test -f $1.1.gz)
then less $1.1.gz
else echo "dont working"
fi
```

```
[ahmet@localhost Lab13]$ bash lab13_2.sh cd
[ahmet@localhost Lab13]$ bash lab13_2.sh cpd
dont working
[ahmet@localhost Lab13]$ bash lab13_2.sh less
```

ahmet@localhost:~/Lab13

File Edit View Search Terminal Help

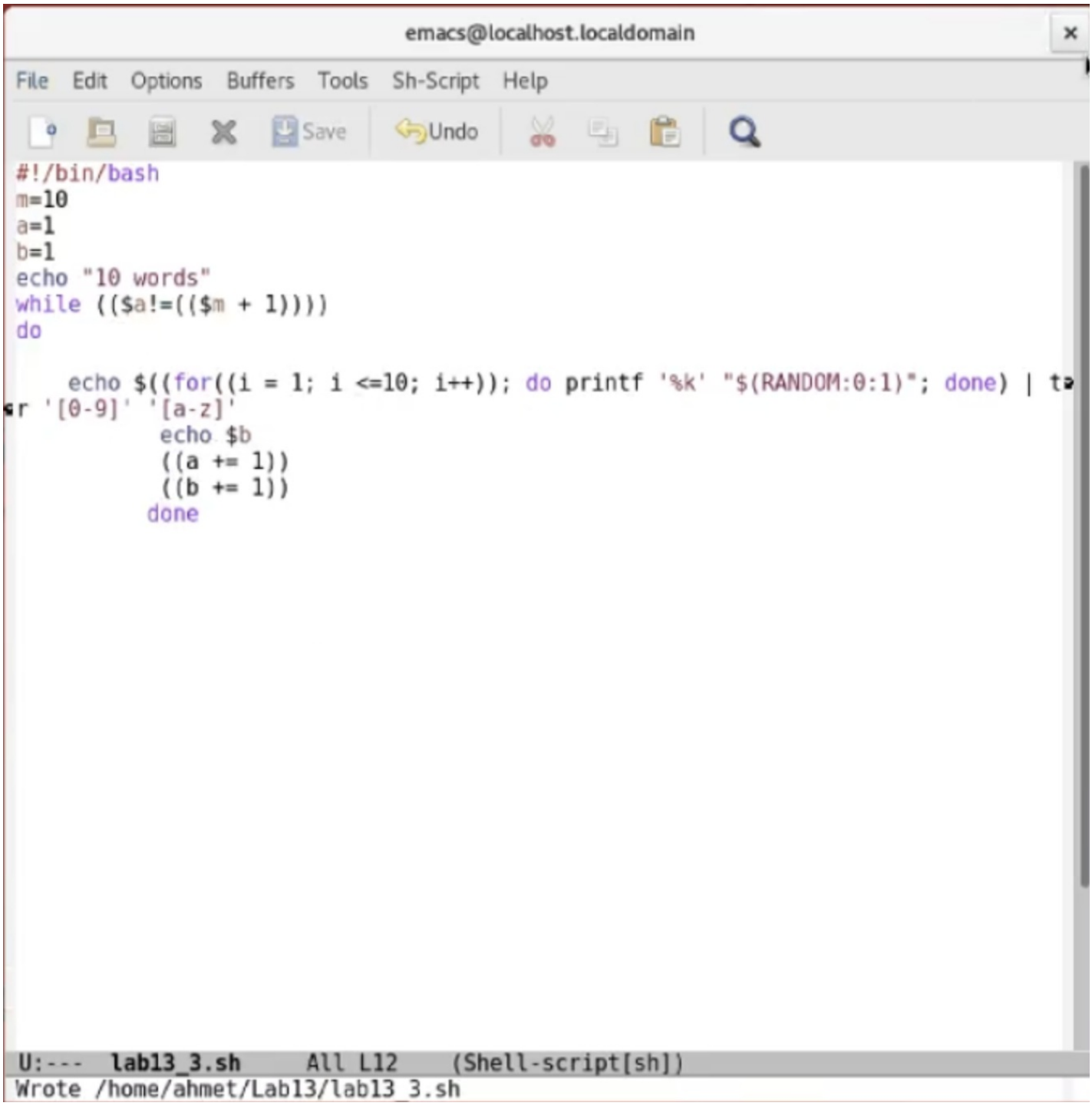
```
BASH_BUILTINS(1)            General Commands Manual            BASH_BUILTINS(1)

ESC[1mNAMEESC[0m
       bash,  :,  .,  [, alias, bg, bind, break, builtin, caller, cd, command,
       compgen, complete, compopt,  continue,  declare,  dirs,  disown,  echo,
       enable,  eval,  exec, exit, export, false, fc, fg, getopts, hash, help,
       history, jobs, kill, let, local, logout, mapfile, popd, printf,  pushd,
       pwd,  read, readonly, return, set, shift, shopt, source, suspend, test,
       times, trap, true, type, typeset, ulimit, umask, unalias, unset, wait -
       bash built-in commands, see ESC[1mbashESC[22m(1)

ESC[1mBASH BUILTIN COMMANDSESC[0m
       Unless otherwise noted, each builtin command documented in this section
       as accepting options preceded by ESC[1m- ESC[22maccepts ESC[1m-- ESC[22mt
o signify the end of the
       options.   The  ESC[1m:ESC[22m, ESC[1mtrueESC[22m, ESC[1mfalseESC[22m, an
d ESC[1mtest ESC[22mbuiltins do not accept options
       and do not treat ESC[1m-- ESC[22mspecially.  The ESC[1mexitESC[22m, ESC[1
mlogoutESC[22m, ESC[1mreturnESC[22m,  ESC[1mbreakESC[22m,  ESC[1mcon-ESC[0m
       ESC[1mtinueESC[22m,  ESC[1mletESC[22m,  and ESC[1mshift ESC[22mbuiltins a
ccept and process arguments beginning
       with ESC[1m- ESC[22mwithout requiring ESC[1m--ESC[22m.  Other builtins th
at accept arguments  but
 ESC
```

3. Используя встроенную переменную *$RANDOM*, написал командный файл, генерирующий случайную последовательность букв латинского алфавита. Учел, что *RANDOM* выдаёт псевдослучайные числа в диапазоне от 0 до 32767. Запустил командный файл. Как видим, вывел рандомные 10 слов, состоящих из рандомных букв латинского алфавита.

```
[ahmet@localhost Lab13]$ touch lab13_3.sh
[ahmet@localhost Lab13]$ chmod +x lab13_3.sh
[ahmet@localhost Lab13]$ emacs
```




```
[ahmet@localhost Lab13]$ bash lab13_3.sh
10 words
lab13_3.sh: line 9: unexpected EOF while looking for matching `)'
lab13_3.sh: line 16: syntax error: unexpected end of file
[ahmet@localhost Lab13]$ emacs
```

Вывод

Изучил основы программирования в оболочке ОС UNIX, научился 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 (коей Bash не содержит). Поддержка для структур данных «хэш». Есть также ряд особенностей, которые присутствуют только в Bash: Опция командной строки –norc, 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 exec, 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, icc, ...) 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 ack(5,2,3)