

# Шаблон отчёта по лабораторной работе № 13

## Операционные Системы

Яссин мохамад аламин

### Содержание

Цель работы.....	1
Задание.....	1
Выполнение лабораторной работы.....	2
Выводы.....	4
Контрольные вопросы.....	4

### Цель работы

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

### Задание

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.

- 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.

## Выполнение лабораторной работы

- Написал командный файл, реализующий упрощённый механизм семафоров.(рис. -@fig:001)

```
#!/bin/bash
lockfile = "/lock_file"
exec {fn}>lockfile

echo "Locked"
until flock -n ${fn}
do
echo "Not locked"
sleep 1
flock -n ${fn}
done
for ((i=0; i<=7; i++))
do
echo "work"
sleep 1
done
```

{ #fig:001 width=70% }

```
[MohamadAlaminYassin@Mohamad ~]$ ./lab13.sh
./lab13.sh: line 2: lockfile: command not found
Locked
work
work
work
work
work
work
work
work
work
```

{ #fig:002 width=70% }

- Реализовал команду man с помощью командного файла. Изучил содержимое каталога /usr/share/man/man1.(рис. -@fig:003)

```
#!/bin/bash
# /usr/share/man/man1
if (test -f $1.gz)
then test $1.gz
else echo "Not working"
fi
```

{ #fig:003 width=70% }

```
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[22mto signify
  the end of the options.
  The ESC[1m:ESC[22m, ESC[1mtrueESC[22m, ESC[1mfalseESC[22m, and ESC[1mtest ESC[22mbu
  iltins do not accept options and do not treat ESC[1m--ESC[0m
  specially. The ESC[1mexitESC[22m, ESC[1mlogoutESC[22m, ESC[1mreturnESC[22m, ESC[1
  mbreakESC[22m, ESC[1mcontinueESC[22m, ESC[1mletESC[22m, and ESC[1mshift ESC[22mbuiltins
  accept and process arguments beginning with ESC[1m- ESC[22mwithout requiring
  ESC[1m--ESC[22m. Other
  builtins that accept arguments but are not specified as accepting options inter-
  pret arguments beginning with ESC[1m- ESC[22mas invalid options and require ESC[1
  m-- ESC[22mto prevent
  this interpretation.
  ESC[1m: ESC[22m[ESC[4margumentsESC[24m]
    No effect; the command does nothing beyond expanding ESC[4margumentsESC[24
  m and per-
    forming any specified redirections. The return status is zero.

ESC[1m. ESC[4mESC[22mfilenameESC[24m [ESC[4margumentsESC[24m]
cd .1.gz
```

```
#!/bin/bash
for symbol in {A..Z} {a..z}
do SYMBOLS=$SYMBOLS$symbol;
done
STR_LEN=20
STR=""
for i in $(seq 1 $STR_LEN)
do
STR=$STR${SYMBOLS:${expr $RANDOM % ${#SYMBOLS}}:1}
done
echo $STR
```

{ #fig:004 width=70% }  
width=70% }

{ #fig:005

1. 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:001)

```

[MohamadAlaminYassin@Mohamad ~]$ mcedit lab13_3.sh

[MohamadAlaminYassin@Mohamad ~]$ ./lab13_3.sh
ocy
[MohamadAlaminYassin@Mohamad ~]$ ./lab13_3.sh
mLH
[MohamadAlaminYassin@Mohamad ~]$ ./lab13_3.sh
TxI
[MohamadAlaminYassin@Mohamad ~]$ ./lab13_3.sh
hYu
[MohamadAlaminYassin@Mohamad ~]$ ./lab13_3.sh
njN
[MohamadAlaminYassin@Mohamad ~]$ ./lab13_3.sh
aXN
[MohamadAlaminYassin@Mohamad ~]$ ./lab13_3.sh
Jgg
[MohamadAlaminYassin@Mohamad ~]$ ./lab13_3.sh
PsL
[MohamadAlaminYassin@Mohamad ~]$ ./lab13_3.sh
ncS
[MohamadAlaminYassin@Mohamad ~]$ ./lab13_3.sh

```

{ #fig:006 width=70% }

## Выводы

В результате работы, я изучил основы программирования в оболочке ОС UNIX. Научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов

## Контрольные вопросы

1. \$1 следует внести в кавычки.
2. С помощью знака >| можно объединить несколько строк в одну.
3. Эта утилита выводит последовательность целых чисел с заданным шагом. Также можно реализовать с помощью утилиты jot.
4. Результатом вычисления выражения  $\$((10/3))$  будет число 3.
5. В zsh можно настроить отдельные сочетания клавиш так, как вам нравится. Использование истории команд в zsh ничем особенным не отличается от bash. Zsh очень удобен для повседневной работы и делает добрую половину рутины за вас. Но стоит обратить внимание на различия между этими двумя оболочками. Например, в zsh после for обязательно вставлять пробел, нумерация массивов в zsh начинается с 1, чего совершенно невозможно понять. Так, если вы используете shell для повседневной работы, исключаящей

написание скриптов, используйте zsh. Если вам часто приходится писать свои скрипты, только bash! Впрочем, можно комбинировать.

6. Синтаксис конструкции `for ((a=1; a <= LIMIT; a++))` верен.
7. Язык bash и другие языки программирования:
  - a. Скорость работы программ на ассемблере может быть более 50% медленнее, чем программ на си/си++, скомпилированных с максимальной оптимизацией;
  - b. 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;
  - c. 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;
  - d. Скорость кодов, генерируемых компилятором языка си фирмы Intel, оказалась заметно меньшей, чем компилятора GNU и иногда LLVM;
  - e. Скорость ассемблерных кодов x86-64 может меньше, чем аналогичных кодов x86, примерно на 10%;
  - f. Оптимизация кодов лучше работает на процессоре Intel;
  - g. 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;
  - h. 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;

- i. 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  $\text{ack}(5,2,3)$