

**Лабораторная работа
№12. Программирование в командном
процессоре ОС UNIX. Расширенное
программирование.**

Операционные системы

Кочарян Никита Робертович

Содержание

1	Цель работы	5
2	Задания	6
3	Выполнение лабораторной работы	8
4	Контрольные вопросы	12
5	Ответы на контрольные вопросы	13
6	Вывод	16

Список иллюстраций

3.1	рис1	8
3.2	рис2	9
3.3	рис3	9
3.4	рис4	10
3.5	рис5	10
3.6	рис6	11
3.7	рис7	11
3.8	рис8	11

Список таблиц

1 Цель работы

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

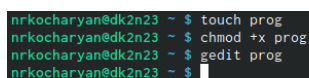
2 Задания

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 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.

3 Выполнение лабораторной работы

1. Пишу 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. Дорабатываю программу так, чтобы имела возможность взаимодействия трёх и более процессов

A screenshot of a terminal window showing four lines of commands and their outputs. The prompt is 'nrkocharyan@dk2n23 ~'. The commands are: 'touch prog', 'chmod +x prog', 'gedit prog', and a blank line followed by a cursor.

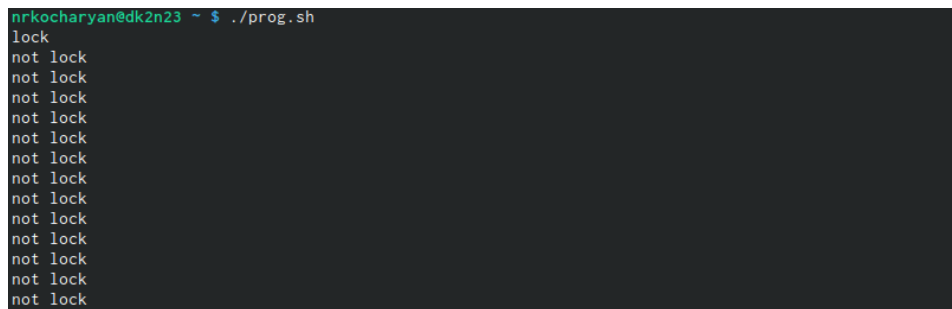
```
nrkocharyan@dk2n23 ~ $ touch prog
nrkocharyan@dk2n23 ~ $ chmod +x prog
nrkocharyan@dk2n23 ~ $ gedit prog
nrkocharyan@dk2n23 ~ $
```

Рис. 3.1: рис1

A screenshot of a GNU Emacs editor window titled "prog - GNU Emacs at dk2n23". The window contains a shell script with the following content:

```
#!/bin/bash
lockfile=".lockfile"
exec {fn}>$lockfile
echo "lock"
until flock -n ${fn}
do
    echo "not lock"
    sleep 1
    flock -n ${fn}
done
do
    echo "work"
    sleep 1
done
```

Рис. 3.2: рис2

A screenshot of a terminal window showing the execution of the script from Figure 3.2. The prompt is "nrkocharyan@dk2n23 ~ \$./prog.sh". The output shows "lock" followed by several "not lock" messages, indicating that the script is running in a loop where it fails to acquire the lock.

```
nrkocharyan@dk2n23 ~ $ ./prog.sh
lock
not lock
not lock
not lock
not lock
not lock
not lock
not lock
not lock
not lock
not lock
not lock
not lock
not lock
not lock
not lock
```

Рис. 3.3: рис3

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`.

```
nrkocharyan@dk2n23 ~ $ touch prog2.sh
nrkocharyan@dk2n23 ~ $ chmod +x prog2.sh
nrkocharyan@dk2n23 ~ $
```

Рис. 3.4: рис4

```
411toppm User Manual(0) 411toppm User Manual(0)
NAME
    411toppm - convert Sony Mavica .411 image to PPM

SYNOPSIS
    411toppm [-width width] [-height height] [411file]

DESCRIPTION
    This program is part of Netpbm(1).

    411toppm reads a .411 file, such as from a Sony Mavic camera, and converts it to a PPM image as output.

    Output is to Standard Output.

    The originator of this program and decipherer of the .411 format, Steve Allen <sla@alumni.caltech.edu>, has
    this to say about the utility of this program: "There's so little image in a 64x48 thumbnail (especially
    when you have the full size JPG file) that the only point in doing this was to answer the implicit chal-
    lenge posed by the manual stating that only the camera can use these files."

OPTIONS
    In addition to the options common to all programs based on libnetpbm (most notably -quiet, see
    Common Options (index.html#commonoptions) ), 411toppm recognizes the following command line options:

    All options may be abbreviated to the shortest unique prefix.

    -width The width (number of columns) of the input image. Default is 64.

    -height
        The height (number of rows) of the input image. Default is 48.

SEE ALSO
    ppm(1)

DOCUMENT SOURCE
    This manual page was generated by the Netpbm tool 'makeman' from HTML source. The master documentation is
    at

    http://netpbm.sourceforge.net/doc/411toppm.html

netpbm documentation 03 March 2001 411toppm User Manual(0)
~
~
~
~
~
~
~
411toppm.1.bz2 lines 1-41/41 (END) - Next: 7z.1.bz2
~:prog x ~:prog x man1:prog2.sh x
```

Рис. 3.5: рис5

- 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.

```
nrkocharyan@dk2n23 ~ $ touch prog3.sh
nrkocharyan@dk2n23 ~ $ chmod +x prog3.sh
nrkocharyan@dk2n23 ~ $
```

Рис. 3.6: рис6

```
Открыть  prog3.sh  Сохранить  ⌵  ⌶  ✕
1 #!/bin/bash
2 M=10
3 c=1
4 d=1
5 echo
6 echo "10 random words:"
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
14
```

Рис. 3.7: рис7

```
nrkocharyan@dk2n23 ~ $ ./prog3.sh
10 random words:
ccbcdbbbbc
1
cbdccicbbc
2
dbbcdhbbbd
3
jccbbbcbed
4
fdfccibccb
5
bbgcgiicbb
6
icbbdbbbdj
7
cfcbcdcgbd
8
bicddbcbbc
9
bdbcccbegc
10
```

Рис. 3.8: рис8

4 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке
`while [$1 != "exit"]`
2. Как объединить (конкатенация) несколько строк в одну?
3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать её функционал при программировании на `bash`?
4. Какой результат даст вычисление выражения `$((10/3))`?
5. Укажите кратко основные отличия командной оболочки `zsh` от `bash`.
6. Проверьте, верен ли синтаксис данной конструкции
`for ((a=1; a <= LIMIT; a++))`
7. Сравните язык `bash` с какими-либо языками программирования. Какие преимущества у `bash` по сравнению с ними? Какие недостатки?

5 Ответы на контрольные вопросы

1. В строке `while [$1 != "exit"]` квадратные скобки надо заменить на круглые.
2. Есть несколько видов конкатенации строк. Например, `VAR1="Hello," VAR2="World" VAR3="VAR1VAR2" echo "$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, awk (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 ask(5,2,3)

6 Вывод

В ходе выполнения данной лабораторной работы я научился расширенному программированию в командном процессоре ОС UNIX,.