Operative Systems [1]

Workbook

Luca Parolari[2]

June 10, 2018

[1] Operative Systems course at UNIPR teached by *Alessandro Dal Palù* (alessandro.dalpalu@unipr.it)
[2] luca.parolari@studenti.unipr.it

Contents

Preface

This paper will be the **workbook** where I solve and document exercises on system programming, shell, threads and so on. The exercises are take from slides seen at lesson, and I'll integrate some pratical and theorical aspects to enrich the content.

The second goal of this paper is to be a little "manual" for some unix's commands and system progrmaming details.

Why in english?

In a world where english is the first international language, I think that doing some practise with it it's a good thing. Futhermore, english is the language to understand and be understood, and I want that my work can be understood by all who wants to read it.

I apologize in advance if my english is not correct or if it sounds bad.

Exercises

Each lesson was discussed presenting a slide, thit contains theorical and pratical aspetcs on the topic of the day and some exercises. You can find the docs on lessons folder.

Structure of document

Each unit will focus on a particular exercise or group of exercises done at lesson but I'll try to integrate with theory hints.

The theory concept are taken from my notes, book and, if necessary, from the internet (sources will be listed in sitography).

Useful links

- Operative System course on Elly at UNIPR.

1

Introduction

Laboratory: Lesson 1

– 10/04/2018

This chapter will talk about fundamental pratical concepts of Linux/Unix OS. In particular, i will talk about linux shell's types and characteristics. Then i will proceed with popular and useful commands like `ls`, `cd`, `man`, `ps`, and many others. After that i will show an overview of Unix file structure and organization and little description for file permissions.

1.1 Shell

The shell is a characters interface. The user can interact with it by executing commands and the shell responds to the user with some messages.

There are many types of shell:

- **sh**: Bourne shell.
- **bash**: Bourne again shell.
- **csh**: C shell.
- **tcsh**: Teach C shell.
- **ksh**: Korn shell.

All the shells provides a programming language, that is very more powerful compared to visual interaction.

1.2 Commands

A command is something that the machine must execute. Commands has this synopsis: `command [options] <arguments>`.

Here there are some commands:

- `date`, shows current date.

```
foo@vtest:~$ date
mar 10 apr 2018, 21.12.38, CEST
```

- `who`, shows users connected to the system.

```
foo@vtest:~$ who
foo  tty2  2018-04-10 16:47 (:1)
```

- `uname -a`, shows system informations.

```
foo@vtest:~$ uname -a
Linux pc-bcvdev 4.13.0-38-generic #43-Ubuntu SMP Wed Mar 14 15:20:44 UTC 2018
    x86_64 x86_64 x86_64 GNU/Linux
```

- `ps`, shows user's processes

```
foo@vtest:~$ ps
PID TTY          TIME CMD
5154 pts/0   00:00:00 bash
12646 pts/0   00:00:00 ps
```

- `ps ef`, shows all processes

```
foo@vtest:~$ ps ef
PID TTY      STAT   TIME COMMAND
5154 pts/0   Ss     0:00 bash [...]
12688 pts/0  R+     0:00  \_ ps ef
1678 tty2    Ssl+   0:00 /usr/lib/gdm3/gdm-x-session
1772 tty2    Sl+    0:00  \_ /usr/lib/gnome-session/gnome-session-binary
1907 tty2    Rl+   19:58     \_ /usr/bin/gnome-shell
1944 tty2    Sl     1:22     |   \_ ibus-daemon
1950 tty2    Sl     0:00     |   |   \_ /usr/lib/ibus/ibus-dconf
2221 tty2    Sl     0:25     |   |   \_ /usr/lib/ibus/ibus-..
6524 tty2    Sl+    4:59     |   \_ gummi
10948 tty2   Sl+    0:04     |   \_ /usr/bin/emacs25

[MORE]
```

This command shows lots of more informations and lots of more processes but these are omitted for clearity.

1.3 Files

1.3.1 Folder structure

Unix usually has standard and repetitive files structure which includes:

Table 1.1: Folder structure

Name	Description
/	Root directory
bin	User commands
sbin	Admin commands
dev	I/O devices
etc	Configuration files
lib	Software library
var	Variable dimension files (ex. logs, mailbox..)
usr	Programs and apps
home	Personal directories for users
proc	Dynamic system informations

1.3.2 File permissions

Linux is a multiuser system. There are four user categories: **root**, which has all permissions on all files and **owner** (u), **group** (g), **world** (o) which are regulated by permission rules.
Every file has an information made of *10 bits* where:

- *the first bit* represents the file type (*file* or *directory*);
- bits *from 2 to 4* represent owner permissions;
- bits *from 5 to 7* represent group permissions;
- bits *from 8 to 10* represent world permissions;

where in this group the first bit is for *read*, the second for *write* and the third for *execute* permission.

1.4 Exercises

1. What is your home directory?
   My home directory is located in `/home/foo/` .

2. Show home directory file ordered by modification date.

```
foo@vtest:~$ ls --sort=t -a -l
totale 177564
drwxr-xr-x 13 foo   foo       4096 apr 10 18:44 Scaricati
drwxr-xr-x 38 foo   foo       4096 apr 10 18:34 .config
drwxr-xr-x 52 foo   foo       4096 apr 10 16:47 .
-rw-------  1 foo   foo      47206 apr 10 16:47 .ICEauthority
-rw-------  1 foo   foo      43169 apr 10 11:29 .bash_history
lrwxrwxrwx  1 foo   foo         23 apr 10 10:54 omega -> /media/foo/Omega/
drwxrwxr-x  5 foo   foo       4096 apr 9 22:14 VirtualBox VMs
drwx------ 38 foo   foo       4096 apr 9 18:34 .cache
drwx------  2 foo   foo       4096 apr 8 19:02 .gconf
drwxr-xr-x 985 foo  foo      36864 apr 6 15:30 .npm
-rw-------  1 foo   foo          0 apr 6 15:26 .node_repl_history
-rw-------  1 foo   foo        537 apr 4 20:13 .xdvirc
drwxr-xr-x 24 foo   foo       4096 mar 31 15:29 .gimp-2.8
drwxr-xr-x  5 foo   foo       4096 mar 30 18:04 .android
drwxr-xr-x  3 foo   foo       4096 mar 30 10:23 Scrivania
drwxr-xr-x  4 foo   foo       4096 mar 28 12:50 .wine
drwxr-xr-x  2 foo   foo       4096 mar 28 12:50 .swipl-dir-history
drwxr-xr-x  2 foo   foo       4096 mar 28 08:37 Video
-rw-r--r--  1 foo   foo        502 mar 27 16:16 .emacs
drwx------  5 foo   foo       4096 mar 26 22:26 .emacs.d
-rw-------  1 foo   foo         34 mar 26 21:18 .lesshst
-rw-------  1 foo   foo         71 mar 24 14:23 .sqlite_history
drwxr-xr-x  3 foo   foo       4096 mar 22 16:42 .texlive2017
-rw-r--r--  1 foo   foo        664 mar 22 16:32 texput.log
drwxr-xr-x 10 foo   foo       4096 mar 20 21:36 Documenti
drwxr-xr-x  2 foo   foo       4096 mar 8 15:35 Immagini
drwx------  2 foo   foo       4096 mar 8 15:05 .ssh
-rw-rw-r--  1 foo   foo         85 mar 5 12:04 .gitconfig
-rw-r--r--  1 foo   foo          0 mar 1 17:38 .odbc.ini
-rw-r-----  1 foo   foo          0 feb 28 17:42 .gksu.lock
drwxrwxr-x  3 foo   foo       4096 feb 27 18:55 .m2
drwxrwxr-x  3 foo   foo       4096 feb 27 17:39 shared
```

```
-rw-------  1 foo   foo       1024 feb 22 17:22 .rnd
-rw-rw-r--  1 foo   foo       3360 feb 14 10:37 .pgadmin3
-rw-rw-r--  1 foo   foo        204 feb 14 09:12 .pgadmin...
-rw-rw-r--  1 foo   foo        834 feb 14 09:12 pgadmin.log
-rw-------  1 foo   foo          0 feb 14 09:02 .pgpass
-rw-r--r--  1 foo   foo         73 feb 14 08:52 .selected_editor
-rw-------  1 foo   foo          0 feb 13 22:40 .psql_history
drwx------  2 foo   foo       4096 feb 8 14:39 .remmina
drwxr-xr-x  2 foo   foo       4096 feb 8 14:38 .vnc
drwx------  3 foo   foo       4096 feb 7 21:37 .gnupg
-rw-r--r--  1 root  root    2238078 feb 7 21:08 .-02.cap
-rw-r--r--  1 root  root   10908742 feb 7 21:08 .-01.cap
-rw-r--r--  1 root  root        597 feb 7 21:08 .-01.kismet.csv
-rw-r--r--  1 root  root       3792 feb 7 21:08 .-01.kismet.netxml
-rw-r--r--  1 root  root       4870 feb 7 21:08 .-02.kismet.netxml
-rw-r--r--  1 root  root        585 feb 7 21:08 .-01.csv
-rw-r--r--  1 root  root        686 feb 7 21:08 .-02.csv
-rw-r--r--  1 root  root        604 feb 7 21:08 .-02.kismet.csv
-rw-rw-r--  1 foo   foo       2423 feb 2 14:55 .overgrive.log
-rw-rw-r--  1 foo   foo          5 feb 2 14:54 .overgrive.lock
-rw-------  1 foo   foo         13 gen 30 16:12 .mysql_history
-rw-------  1 foo   foo       8192 gen 26 19:01 lp2s0
-rw-------  1 foo   foo       8192 gen 26 19:01 ls20
drwx------  3 foo   foo       4096 gen 26 08:12 .putty
drwx------  3 foo   foo       4096 gen 25 16:34 .gnome
-rw-rw-r--  1 foo   foo     155054 gen 25 13:31 java...
drwxrwxr-x  3 foo   foo       4096 gen 22 23:06 .tixati
drwxr-xr-x  2 foo   foo       4096 gen 22 22:23 Desktop
drwxr-xr-x  3 foo   foo       4096 gen 22 22:23 snap
drwxrwxr-x  2 foo   foo       4096 gen 22 21:41 MEGAsync
drwx------  3 foo   foo       4096 gen 22 13:58 .thumbnails
drwx------  4 foo   foo       4096 gen 22 13:57 .nv
drwx------  2 root  root      4096 gen 22 13:27 .gvfs
drwxrwxr-x  3 foo   foo       4096 gen 21 12:19 Sviluppo
-rw-r--r--  1 foo   foo        401 gen 21 12:13 id_rsa_20170121.pub
-rw-------  1 foo   foo       1766 gen 21 12:13 id_rsa_20170121
drwxr-xr-x  2 foo   foo       4096 gen 20 20:23 .swt
drwxr-xr-x  4 foo   foo       4096 gen 20 19:51 .smartgit
drwx------  4 foo   foo       4096 gen 20 18:18 .thunderbird
drwx------  3 foo   foo       4096 gen 20 18:14 .mysql
drwx------  3 root  root      4096 gen 20 18:04 .dbus
drwxr-xr-x  4 foo   foo       4096 gen 20 18:02 .IntelliJIdea2017.2
drwxr-xr-x  4 foo   foo       4096 gen 20 17:53 .PhpStorm2017.2
drwxr-xr-x  7 foo   foo       4096 gen 20 17:48 .gradle
-rw-------  1 foo   foo  168026112 gen 20 17:20 core
drwxrwxr-x  3 foo   foo       4096 gen 20 17:20 .vscode
drwx------  3 foo   foo       4096 gen 20 17:20 .pki
drwxr-xr-x  3 foo   foo       4096 gen 20 17:19 Android
drwxr-xr-x  4 foo   foo       4096 gen 20 17:18 .java
drwxr-xr-x  4 foo   foo       4096 gen 20 17:18 .AndroidStudio3.0
drwx------  5 foo   foo       4096 gen 20 15:30 .mozilla
-rw-r--r--  1 foo   foo          0 gen 20 15:28 .s ...
drwxr-xr-x  2 foo   foo       4096 gen 20 14:28 Modelli
drwxr-xr-x  2 foo   foo       4096 gen 20 14:28 Musica
drwxr-xr-x  2 foo   foo       4096 gen 20 14:28 Pubblici
drwxr-xr-x  3 foo   foo       4096 gen 20 14:28 .local
-rw-r--r--  1 foo   foo        220 gen 20 13:21 .bash_logout
-rw-r--r--  1 foo   foo       3771 gen 20 13:21 .bashrc
```

```
-rw-r--r--  1 foo   foo       8980 gen 20 13:21 examples.desktop
-rw-r--r--  1 foo   foo        675 gen 20 13:21 .profile
drwxr-xr-x  3 root  root      4096 gen 20 13:21 ..
```

3. What is the difference between `cat`, `more` and `tail`?
   `Cat` is a program used to concatenate files and print them on standard output.
   `Tail` instead print the last 10 lines of each file to standard output.
   And `more` is a filter for paging through text one screenful at a time.

4. Find a way to show home subdirectories recursively.
   There are more than one way.

   - `ls -R` shows directories recursively;
   - `ls -R | grep ":$"| sed -e 's/:$//'-e 's/[^-][^\/]*\//--/g'-e 's/^/ /'-e 's/-/|/'` found on google, manipulates the output of command described before and output something like this:

```
|-Android
|---Sdk
|-----build-tools
|-------21.1.2
|---------lib
|---------renderscript
|-----------clang-include
|-----------include
|-----------lib
|-------------bc
|---------------armeabi-v7a
|---------------mips
|---------------x86
|-------------intermediates
|---------------armeabi-v7a
|---------------mips
|---------------x86
|-------------packaged
|---------------armeabi-v7a
|---------------mips
|---------------x86
|-------26.0.2
|---------lib
|---------lib64
|---------renderscript
|-----------clang-include
|-----------include
|-----------lib

[MORE]
```

   - `tree` (installed with `sudo apt-get install tree`) shows something like this:

```
+-- Android
|   +-- Sdk
|       +-- build-tools
|       |   +-- 21.1.2
|       |   |   +-- aapt
|       |   |   +-- aidl
```

```
|       |   |   +-- arm-linux-androideabi-ld
|       |   |   +-- bcc_compat
|       |   |   +-- dexdump
|       |   |   +-- dx
|       |   |   +-- i686-linux-android-ld
|       |   |   +-- jack.jar
|       |   |   +-- jill.jar
|       |   |   +-- lib
|       |   |   |   +-- dx.jar
|       |   |   |   +-- shrinkedAndroid.jar
|       |   |   +-- libbcc.so
|       |   |   +-- libbcinfo.so
|       |   |   +-- libclang.so
|       |   |   +-- libc++.so
|       |   |   +-- libLLVM.so
|       |   |   +-- llvm-rs-cc
|       |   |   +-- mainDexClasses
|       |   |   +-- mainDexClasses.rules
|       |   |   +-- mipsel-linux-android-ld
|       |   |   +-- NOTICE.txt
|       |   |   +-- package.xml
|       |   |   +-- renderscript

[MORE]
```

5. What following commands do?

```
1: > cd
2: > mkdir d1
3: > chmod 444 d1
4: > cd d1
```

At *line 1* the current dir is setted to home (tilde); at *line 2* a new directory called `d1` is created and then, at *line 3* it's permissions are setted. The permissions say that all can read but nobody can write or execute the directory. At *line 4* is returned an error which says that we don't have permission to "run" (i.e. access) the directory.

2

Bash

Laboratory: Lesson 2

– 17/04/2018

2.1 Exercises

1) Write an unique command (pipeline) for:

   a) copying the content in dir1 to dir2;

   b) showing number of files contained in directories recursively (use `ls -R` and `find`);

   c) showing home's directory files which star with 3 chars and a number.

   Solution.

   a) `cp -r dir1/* dir2`, copies recursively the content of `dir1` to `dir2`

   b) `ls -R | find | wc -l`, shows the number of lines produced by find command. The first command executed is `ls -R` which outputs

```
foo@vtest:~/Documents/es/studenti$ ls -R
.:
dir1  dir2  f1.txt  f2.txt

./dir1:
f1.txt  f2.txt

./dir2:
f1.txt  f2.txt
```

   then `find` that outputs

```
foo@vtest:~/Documents/es/studenti$ ls -R | find
.
./f2.txt
./f1.txt
./dir1
./dir1/f2.txt
./dir1/f1.txt
./dir2
./dir2/f2.txt
./dir2/f1.txt
```

   and finally `wc -l` which counts number of lines (i.e. numer of files and directory)

```
foo@vtest:~/Documents/es/studenti$ ls -R | find | wc -l
9
```

c) `echo /[a-z][a-z][a-z][0-9]*`

2) What are the difference between:

a) `ls`;

b) `ls | cat`;

c) `ls | more`.

Solution.

a) `ls` list information about files in a directory;

b) `ls | cat` takes the output of `ls` and prints it on stdout;

c) `ls | more` takes the output of `ls` and prints with `more` format, providing filters and interactivity.

3) What are the effects of following commands?

a) `uniq < file`;

b) `who | wc -l`;

c) `ps -e | wc -l`.

Solution.

a) `uniq < file`, compares adjacent lines and merge to the first occurrence matching lines in $file$eq;

b) `who | wc -l`, prints the number of lines from *input*. In this case input is `who` which show logged in users, so `who | wc -l` shows number of logged users;

c) `ps -e | wc -l`, like the previos but here `ps -e | wc -l` shows number of processes running in the system.

4) Override `rm` command so that no delete confirmation is asked.

Solution.

```
foo@vtest:~$ alias rm="rm --interactive=never"
foo@vtest:~$ alias rmi="rm -i"
```

5) Write a pipeline which outputs number of executing process.

Solution.

```
foo@vtest:~/Documents$ ps -e | wc -l
```

6) Save to a file last command with `ls`.

Solution.

```
foo@vtest:~/Documents$ !ls > t.txt
```

7) Write a command which shows number of commands saved in history.

Solution.

```
foo@vtest:~/Documents$ history | wc -l
```

8) Write a command which shows top 15 commands saved in history.

Solution.

```
foo@vtest:~/Documents$ history | tail -n 15
```

9) What are unix commands which start with *lo*?.

Solution.

```
foo@vtest:~$ lo[TAB]
loadkeys      locate         logout         lorder
loadunimap    lodraw         logresolve     losetup
local         loffice        logrotate      loweb
localc        lofromtemplate logsave        lowntfs-3g
locale        logger         loimpress      lowriter
localectl     login          lollipop
localedef     loginctl       lomath
locale-gen    logname        look
```

10) Create at least 2 ways to display home dir files starting with *al*?.

Solution.

```
foo@vtest:~$ ?????
```

11) What is the result of following commands?.

a) `ls -R || (echo file non accessibili > tmp)`

b) `(who | grep rossi)&& cd ~rossi`

c) `(cd / ; pwd ; ls | wc -l )`

Solution.

a) `(echo file non accessibili > tmp)` will be executed only if `ls -R` fails, if everything it's fine `ls -R` will outputted on screen;

b) if `(who | grep rossi)` is successful (rossi is logged in), change directory to `~rossi`

c) `(cd / ; pwd ; ls | wc -l )` moves to root directoy, show on screen *present working directory* then shows number of files and directories in /.

3

More on Bash, Filters

Laboratory: Lesson 3

– 08/05/2018

3.1 Exercises

1) Execute `(xterm&); (nice xterm&); (nice -n 19 xterm&); ps -el`

Solution.

```
foo@vtest:~$ (xterm&) ; (nice xterm&) ; (nice -n 19 xterm&); ps -el
0 R  1000 13190 1852 0  80   0 - 20376 -    pts/1   00:00:00 xterm
0 S  1000 13192 1852 0  90  10 - 20376 poll_s pts/1 00:00:00 xterm
0 R  1000 13194 1852 0  99  19 - 16758 -    pts/1   00:00:00 xterm
```

2) What is the effect of `sort file > file`?

Solution. The output of sort command is redirected and writed on file.

3) What is the effect of `tr str1 str2` if $|str1| \neq |str2|$?

Solution. *tr* translates a text by $str1[i]$ occurence with $str2[i]$. If $str2 < str1$ the the last charater of $str2$ is repeated as necessary. If $str2 \geq str1$ normal transaltion of $str1$ happens.

4) Replace alphanumerics characters with a <tab>.

Solution. `cat b.txt | tr -s A-Za-z0-9 \\t`

5) Write a pipeline that discover repeated lines in a file.

Solution. `cat b.txt | sort | uniq --count`

6) Write a command to discover users that has at least a process running on the system and output them once.

Solution.

```
foo@vtest:~/temp/t$ ps -fe | awk ' { if ($1!="UID") { print $1 } } ' | sort |
    uniq
avahi
colord
foo
gdm
kernoops
```

```
message+
mysql
nvidia-+
postgres
root
rtkit
syslog
systemd+
whoopsie
www-data


foo@vtest:~/temp/t$ ps -f | awk ' { if ($1!="UID") { print $1 } } ' | sort | uniq
foo
```

7)
```
foo@vtest:~$ sed '4,$d' passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

8)
```
foo@vtest:~$ sed '3q' passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

9) Without sed the output is

```
foo@vtest:~$ cat passwd | grep root
root:x:0:0:root:/root:/bin/bash
```

with sed instead

```
foo@vtest:~$ sed /sh/y/:0/_%/ passwd | grep root
root_x_%_%_root_/root_/bin/bash
```

This sed script triggers when *sh* string is found, than with *y* it translates : 0 in _% respectively.

10) Without sed the output is

```
foo@vtest:~$ cat passwd | grep mysql
mysql:x:123:129:MySQL Server,,,:/nonexistent:/bin/false
```

with sed instead

```
foo@vtest:~$ sed '/sh/!y/:0/_%/' passwd | grep mysql
mysql_x_123_129_MySQL Server,,,_/nonexistent_/bin/false
```

This sed script triggers when *sh* string is not found, than with *y* it translates : 0 in _% respectively.

11) Without sed the output is

```
foo@vtest:~$ cat passwd | grep root
root:x:0:0:root:/root:/bin/bash
```

with sed `disabled.txt` contains

```
foo@vtest:~$ sed '/^root/,/^bin/s/:.....:/::/w disabled.txt' passwd


root:x::/root:/bin/bash
daemon:x:1::/usr/sbin:/usr/sbin/nologin
bin:x:2:2::/usr/sbin/nologin
```

12) Without sed the output is

```
foo@vtest:~$ cat passwd | grep root
root:x:0:0:root:/root:/bin/bash
```

with sed instead

```
foo@vtest:~$ cat passwd | sed 's?/bin/.*sh$?/usr/local&?' | grep root
root:x:0:0:root:/root:/usr/local/bin/bash
```

13) Write a command with awk that prints the max lenght of a line in a file.

Solution. I created a file with random words per line

```
foo@vtest:~$ sed '/$/y/:/ /' passwd > passwd_div.txt
```

This command replaces all : with *space*.

The awk script is

```
BEGIN { print "Searching the longest line..." }
{
    if (NF>max) {
   max=NF;
   line=NR;
    }
}
END { printf "The longest line number is %d, it has %d words.\n", line, max }
```

and it's output

```
foo@vtest:~$ awk -f es_awk.sh passwd_div.txt
Searching the longest line...
The longest line number is 17, it has 10 words
```

4

Scripts

Laboratory: Lesson 4

– 22/05/2018

4.1 Bash' scripts

Shell' scripts are intepreted by shell and they are writed with atomics commands. The script can be executed if it has the x permission, and the firt line must be `#!/bin/bash` to make the bash shell interpret commands. Scripts are executed writing `./script_name` on the shell.

A script can interact with I/O throught `read` and `echo` instructions.

Note: backquote is an important function of scripts: it allows to take the stdout of a command and store it in a variable.

4.2 Statements

A script has a basic syntax for programming language but the common statements are available.

if allows to test a condition, in particular commands' exit status. When the exit status is not available we can can use the command `test`. It provides a lot of parameters to manipulate the test condition.

while allows to iterate until the condition become false.

for allows to iterate in a sequence.

case allows to handle multiple choices.

select builds a question for the user and the select the answered by user.

4.3 Exercises

4.3.1 Draw Triangle

Create a command named *drawtriangle* which prints a triangle with give height.
Solution.

```
#!/bin/bash

# display the usage of drawitriangle and exit with code 1
usage() {
    echo "usage: drawtriangle <height>"
    echo ""
    echo " <height>: the height of the triangle."
    echo "  must be a number"
```

```
    echo "  must be between 3 and 15"
   exit 1
}



height=$1


# test if height is given
if [ $# -ne 1 ]
then usage; fi;

# test if height is a number
re='^[0-9]+$'
if ! [[ $height =~ $re ]] ; then
   usage
fi

# test if 3 < height < 15
if test $height -lt 3 -o $height -gt 15
then usage; fi;


# drawing triangle
for i in `seq 1 $[$height-1]` # iterate on height
do
    echo -n "|"
    for j in `seq 1 $[$i-1]` # iterate on spaces between left and right
    do
    echo -n " "
    done
    echo "\\" # draw right side
done
for i in `seq 1 $[$height+1]` # draw the base
do
    echo -n "-"
done
echo ""
```

The output:

```
foo@vtest:~$ ./drawtriagnle.sh 100
usage: drawtriangle <height>

 <height>: the height of the triangle.
   must be a number
   must be between 3 and 15


foo@vtest:~$ ./drawtriagnle.sh 5
|\
| \
|  \
|   \
------
```

4.3.2 Clear Core

Create a command that delete recursively all files named *core* starting from *dir*.
Solution.

```
#!/bin/bash

# display the usage of clearcore and exit with code 1
usage() {
    echo "usage: clearcore <dirpath>"
    echo ""
    echo "Deletes all file named 'core' recursively from dirpath"
    echo ""
    echo "ARGS:"
    echo " <dirpath>: directory from which deleting core files."
    exit 1
}


dirpath=$1


# test if dirpath is given
if [ $# -ne 1 ]
then usage; fi;

# test if dirpath exist and if dirpath is a directory
if test ! -e $dirpath -o ! -d $dirpath
then usage; fi;


echo "Following file will be deleted:"
echo ""
for i in `find $dirpath -name core -type f`
do
    echo $i
done
echo ""
echo "are you sure?"
select continue in "yes" "no"
do
    for i in `find $dirpath -name core -type f`
    do
      rm $i
    done
    echo ""
    echo "Deleted"
    break
done
```

The output if non valid input

```
foo@vtest:~$ ./clearcore.sh
usage: clearcore <dirpath>

Deletes all file named 'core' recursively from dirpath

ARGS:
 <dirpath>: directory from which deleting core files.
```

and if valid

```
foo@vtest:~$ ./clearcore.sh dir
Following file will be deleted:

dir/core
dir/d1/core

are you sure?
1) yes
2) no
#? yes

Deleted
```

4.3.3 Process Utility

Create a user-friendly process helper that allows to

- giving a user, show pid and command line which has created process,
- show all system's processes orderer by pid,
- $kill - 9$ a process using pid.

Solution.

```
#!/bin/bash

BLUE='\033[0;34m'
NC='\033[0m' # no color

usage() {
    echo ""
    echo "HELP"
    echo ""
    echo " USAGE: processutils.sh"
    echo ""
    echo " ACTIONS:"
    echo "  <h|help>: show this guide"
    echo "  <1|p|pid>: pid and command line giving user"
    echo "  <2|a|all>: show system processes ordered by pid"
    echo "  <3|k|kill>: kill -9 on pid"
    echo "  <4|e|exit>: exit from this tool"
    echo " where | is OR."
    echo ""
    echo " EXAMPLES:"
    echo "  1) this can be a possible sequence of operations: "
    echo "       foo@foo:~$ processutils.sh"
    echo "       Chose an action [p,a,k,e] or h for help: p"
    echo "       Enter the user: user"
    echo "       UID       TTY"
    echo "       lparola+  pts/0"
    echo "     p is the action selected, and than user is the user searched in system"
    echo "  2) or this: "
    echo "       foo@foo:~$ processutils.sh"
    echo "       Chose an action [p,a,k,e] or h for help: e"
    echo "       Exiting..."
    echo "       foo@foo:~$"
```

```
    echo "  3) even this: "
    echo "       foo@foo:~$ processutils.sh"
    echo "       Chose an action [p,a,k,e] or h for help: hello"
    echo "       Unknown action 'hello'"
    echo ""
}

pid() {
    echo ""
    echo "ACTION: PID"
    echo -n "Enter the user: "
    read user
    echo ""
    ps -f | awk ' { printf("%-10s %s\n", $1, $6) } ' #' { if ($1!="UID") { print $1 "
        " $2 } } ' | sort
}

all() {
    echo ""
    echo "ACTION: ALL"
    ps -fe --sort=pid
}

kill_pid() {
    echo ""
    echo "ACTION: KILL"
    echo -n "Enter pid to kill: "
    read pid
    kill -9 $pid
}


echo -e $BLUE"Welcome to Process Utils CLI"
echo "Made by lparolari <luca.parolari23@gmail.com>"
echo -e $NC""

usage

while true
do
    echo "******************"
    echo "Chose an action [p,a,k,e] or h for help:"
    echo -n "#? "


    read action

    case $action in
   "h"|"help")
       usage
       ;;
   1|"p"|"pid")
       pid
       ;;
   2|"a"|"all")
       all
       ;;
   3|"k"|"kill")
```

```
      kill_pid
      ;;
  4|"e"|"exit")
      echo "Exiting..."
      exit 0;;
  *)
      echo "Unknown action '$action'"
   esac

   echo ""
   echo ""
done
```

and some examples:

```
foo@vtest:~$ ./processutils.sh
Welcome to Process Utils CLI
Made by lparolari <luca.parolari23@gmail.com>


HELP

 USAGE: processutils.sh

 ACTIONS:
   <h|help>: show this guide
   <1|p|pid>: pid and command line giving user
   <2|a|all>: show system processes ordered by pid
   <3|k|kill>: kill -9 on pid
   <4|e|exit>: exit from this tool
  where | is OR.

 EXAMPLES:
   1) this can be a possible sequence of operations:
        foo@foo:~$ processutils.sh
        Chose an action [p,a,k,e] or h for help: p
        Enter the user: user
        UID       TTY
        lparola+  pts/0
      p is the action selected, and than user is the user searched in system
   2) or this:
        foo@foo:~$ processutils.sh
        Chose an action [p,a,k,e] or h for help: e
        Exiting...
        foo@foo:~$
   3) even this:
        foo@foo:~$ processutils.sh
        Chose an action [p,a,k,e] or h for help: hello
        Unknown action 'hello'

******************
Chose an action [p,a,k,e] or h for help:
#? p

ACTION: PID
Enter the user: foo

UID   TTY
```

```
foo  pts/0
foo  pts/0
foo  pts/0


******************
Chose an action [p,a,k,e] or h for help:
#? all

ACTION: ALL
UID       PID  PPID  C STIME TTY        TIME CMD
root        1     0  0 16:31 ?      00:00:04 /sbin/init splash
[more]
foo      7616  6563  0 18:28 pts/0  00:00:00 /bin/bash ./processutils.sh
foo      7629  6554  0 18:28 pts/1  00:00:00 bash


******************
Chose an action [p,a,k,e] or h for help:
#? k

ACTION: KILL
Enter pid to kill: 7629


******************
Chose an action [p,a,k,e] or h for help:
#? a

ACTION: ALL
UID  PID  PPID  C STIME TTY    TIME CMD
root        1     0  0 16:31 ? 00:00:04 /sbin/init splash
[more]
foo  7616  6563  0 18:28 pts/0  00:00:00 /bin/bash ./processutils.sh


******************
Chose an action [p,a,k,e] or h for help:
#? e
Exiting...
foo@vtest:~$
```

5

System Programming

Laboratory: Lesson 5

– 29/05/2018

5.1 Fork

5.1.1 Father and Son

```
/*!
\mainpage Esempio Fork
\section intro Introduzione
esempio di creazione processi\n

\date 31/05/2018
\version  0.0.0.1
\author Luca Parolari
*/

/*!
*
* \file     esempiofork.c
* \brief    file principale
* \author   L. Parolari
* \date     31.05.18
*/

#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>

#define PROVA 5 //!< Questa e' una macro globale

int temp; //!< Questa e' una Variabile globale

/********************** main() **********************/
/*!
    \brief Questa e' la funzione principale
    \param argc Il numero di argomenti
    \param argv Il vettore degli argomenti
    \return 0 = Esecuzione terminata con successo
    \return -1 = Esecuzione ha generato un errore
```

```
   Lo scopo di questa funzione e' quello di lanciare un processo figlio
   e attendere la sua terminazione
*/

int main(int argc, char *argv[])
{
  int pid;

  int retv; /// Si utilizza retv per ricevere il valore di ritorno del figlio

  switch(pid=fork()) {
  case -1:
    printf("Errore in creazione figlio\n");
    exit(-1);
  case 0 :
    printf("Figlio sospende per 2 secondi...\n"); // figlio
    printf("Pid del figlio: %d, pid di suo padre: %d\n", getpid(), getppid());
    sleep(5);
    printf("Figlio si risveglia\n");
    exit(3);
  default:
    printf("Padre esegue e attende figlio\n"); // padre
    printf("Pid del padre: %d, pid di suo padre: %d\n", getpid(), getppid());
    wait(&retv);
    printf("Padre riceve da figlio exit status %d\n",WEXITSTATUS(retv));
  }

 return 0;
}
```

and this is the output:

```
foo@vtest:~/es1$ ./esempiofork
Padre esegue e attende figlio
Pid del padre: 19234, pid di suo padre: 19198
Figlio sospende per 2 secondi...
Pid del figlio: 19235, pid di suo padre: 19234
Figlio si risveglia
Padre riceve da figlio exit status 3
```

We can observe that the parent pid of the father is 19198, which is the shell's pid.

Note: this types of comments are added to make *doxygen* works. A small example of generated documentation is:

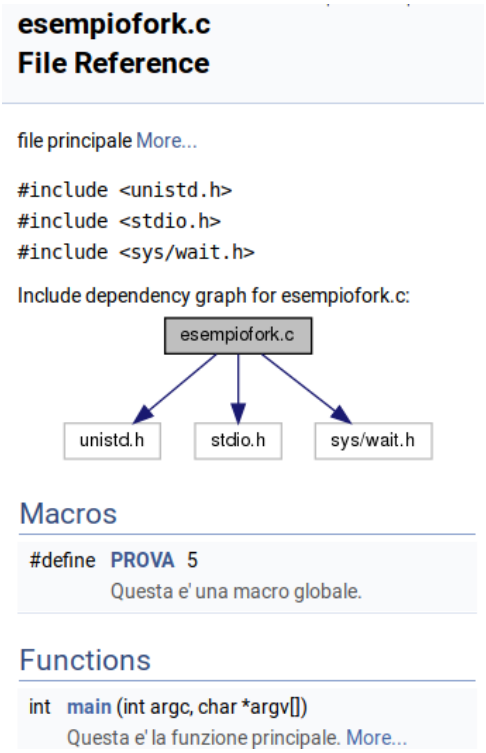

Figure 5.1: Generated docimentation for esempiofork.c file

5.2 Automatic Docs

Documenting a project is a must. All programmers has to document their code in order to interact with others programmer and easily mantain code.

But creating a manual is a complex and long work. To solve this problem there are some tools which can be used to automatically generate the manual. One of this is *Doxygen*. Doxygen uses a configuration file for settings created with `doxygen -g`. With the command `doxygen` executed in target folder it generate manuals in different formats (latex, pdf, html...) reading from all sources file the formatted comments.

Obviously doxygen uses some markers placed in comments in order to generate documentation.

5.3 Files

5.3.1 Read and Skip

This program read from a file 2 characters, prints them on screen and skips the next 3 until the file is empty.

```
/*!
\mainpage Read and Skip
\section intro Introduzione
The goal of this program is to read from a file 2 characters, prints them on screen
    and skips the next 3 until the file is empty.\n

\date 31/05/2018
\version 0.1
\author Luca Parolari
```

```
*/

/*!
* \file     read_and_skip.c
* \brief    Main file
* \author   L. Parolari
* \date     31/05/2018
*/

#define _GNU_SOURCE

#include <stdlib.h>
#include <unistd.h>
#include <limits.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

/********************** main() **********************/
/*!
    \brief Main function.
    \param argc Arguments' number
    \param argv Arguments' values.
    \return 0 = Success
    \return -1 = Error occurred

    The goal of this program is to read from a file 2 characters, prints them on
        screen and skips the next 3 until the file is empty.
*/
int main() {
  int file;

  // opens data.txt.
  file = open( "data.txt", O_RDONLY);
  if(file == - 1) {
    printf( "%s\n", strerror(errno) );
    exit(-1);
  }

  int nread = 0; // number of chars readed.
  int nlseek = 0; // number of chars skipped.
  char data[2];

  while(1) {
    nread = read(file, data, 2);
    nlseek = lseek(file,3,SEEK_CUR);
    if (nread <= 0) break;
    printf( "%s", data);
    //printf( ", nread: %d", nread);
    //printf( ", nlseek: %d\n", nlseek);
  }
  if(nread == -1 || nlseek == -1) {
    printf( "Error(s) occurred: %s\n", strerror(errno));
    exit(-1);
  }

  printf("\n");
```

```
  close(file);

  exit(0);
}

// Input: abcdefghijk (from file data.txt)
// Output: abfgjk
```

5.3.2 Mysh Mod

Modify the simple shell adding the which command and the possibility to execute all commands from down machine.

See chapter 7.

6

Processes and IPC

Linux has different abstractions for synconization and communication:

- Unix pipe: between 2 process throught stream;
- Unix RPC
- Unix Signals
- BSD Socket
- SysV IPC (83) InterProcess Communication
  - Shared memory segments (`shmget()`, `shmctl()`, ...)
  - Semaphores (`semget()`, `semctl()`, ...)
- Pthread:
  - Shared memory between processes
  - Semaphores

Note: `pipe(fd[2])` does not creates shared memory, it creates an unidirectional data channel that can be used for interprocess communication.

6.1 IPC: Inter Process Communication

Every IPC object has an id.

If two process uses the same IPC, they has to share a key. The key is generated with a pathname (for rights) and an id. Every process that can access a directory can request the same key, then, for each directory we can generate 256 different key.
Shared memory is not deleted with process until the system is stoppped.

The function to get a shared memory segment is `int shmget(key, size, flags)` this returns an identified for the shared memory. Then the shared memory needs to be attached to process in order to allow the process to access to this memory (internally, shared memory fragment is added to the process page table).

The IPC can be controlled with the function `int <ipc>ctl (...)`, instanciable on each ipc.

Unix provides some shell commands in order to control IPC:

```
foo@vtest:~$ ipcs
------ Segm. Memoria Condivisa --------
chiave    shmid     proprietario perms    byte      nattch    stato
```

```
0x00000000 131072   274404    700       8110080   2         dest
0x00000000 163841   274404    700       21912     2         dest
0x00000000 65538    274404    700       13860     2         dest
0x00000000 98307    274404    700       184320    2         dest
0x00000000 196612   274404    700       18084     2         dest
0x00000000 229381   274404    700       25476     2         dest
0x00000000 262150   274404    700       27720     2         dest
0x00000000 294919   274404    700       21780     2         dest
0x00000000 327688   274404    600       393216    2         dest
0x00000000 360457   274404    700       23496     2         dest
0x00000000 491530   274404    700       30888     2         dest
0x00000000 524299   274404    700       20196     2         dest
0x00000000 557068   274404    700       29832     2         dest
0x00000000 589837   274404    600       393216    2         dest
0x00000000 655374   274404    700       26532     2         dest
0x00000000 688143   274404    600       393216    2         dest
0x00000000 720912   274404    600       393216    2         dest
0x00000000 753681   274404    700       31548     2         dest
0x00000000 786450   274404    700       22044     2         dest
0x00000000 819219   274404    700       7835520   2         dest
0x00000000 851988   274404    600       393216    2         dest
0x00000000 884757   274404    600       393216    2         dest

------ Matrici semafori --------
chiave    semid     proprietario perms    nsems

------ Code messaggi --------
chiave    msqid     proprietario perms    byte utilizzati messaggi
```

where messages queue are mailboxes.

6.2 Semaphores

Executing operation on semaphores (unix allows to perform more operations atomically on one semaphore):

```
int semop(id, ops[], number_op)}
```

where *ops* can be composed with

- +1 (up)
- -1 (down)
- 0 (sleep if 0)

The function for semaphore control is `int semctl(id, initial_number, arg)`

6.3 Exercises

See chapter 7.

7

Other execises

The other exercises assigned at last laboratory lessons are in the attachments of this document.

In the attachments there are the solutions for:

- Lesson 5
  - page 6, esempiofork;
  - page 16, read and write;
  - page 18, read and skip;
  - pages 20, 21, mysh.

  Something drom lesson 5 is also discussed above.
- Lesson 6
  - page 5, pipe;
  - page 17, philophers.
- Lesson 7
  - page 17, read-writes (db protocol).