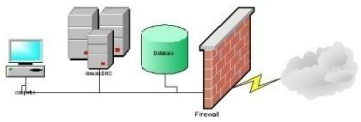
Linux System Admin

- brief overview-

Pál Varga

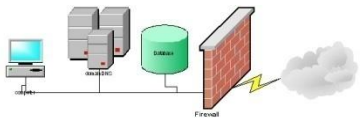
pvarga@tmit.bme.hu



Overview

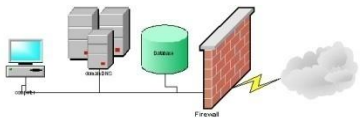
- Overview of the Linux kernel
- File System
- Main operations

- Shell
- Scripting
- General info on the Management of Inf. Systems laboratory

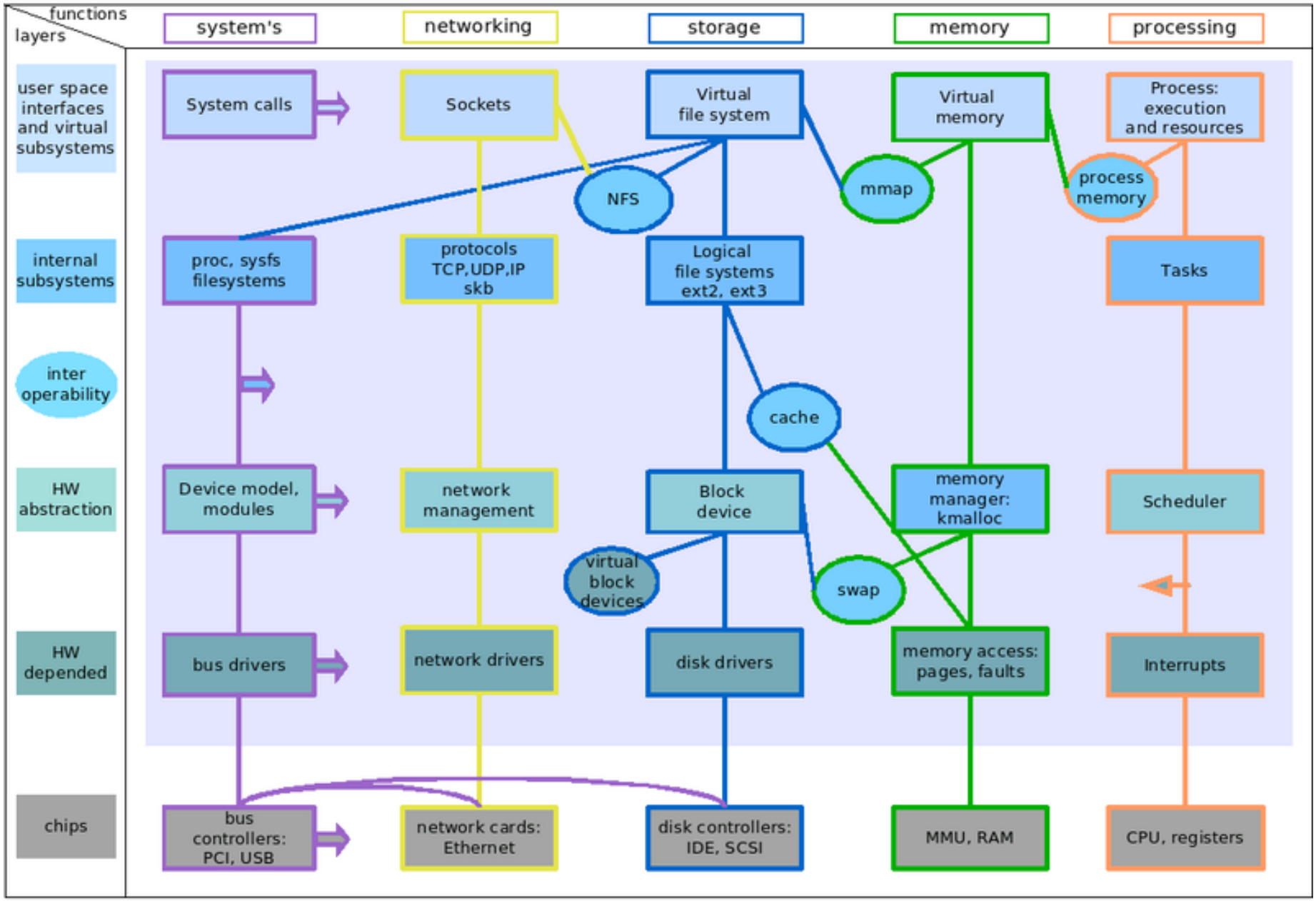


LINUX SYSTEM ADMINISTRATION

I. Kernel
II. File System
III. Main operations



Simplified Linux kernel diagram in form of a matrix map



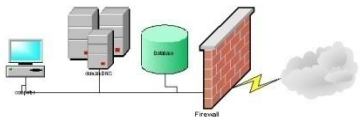
File System

- „device” or „partition” that is formatted for file storage
- Many formats – same appearance for the user: directory structure
- /proc/filesystems - pl. ext2 ext3 vfat NFS ntfs

- To access: *mount* the device: it then appears as a directory

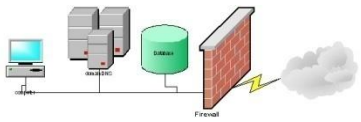
  mount -t vfat /dev/hda1 /windows/cdrive

- mount / umount /eject



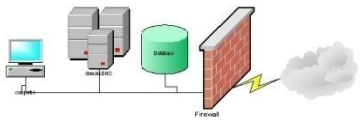
File System (cont)

- Also an application
  - To control data access
  - To handle and access directories
  - Other applications also use its services
    - creates files and directories
    - opens and modifies already existing ones
    - deletion
    - access control



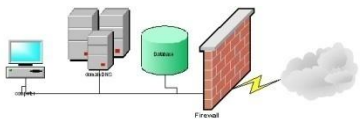
A few well known directories

/bin	common programs, shared by admins, users
/boot	startup files and kernel. Also GRUB data
/dev	references to all hardware as special files
/etc	important system configuration files. Sort of like Control Panel
/home	containes home directories for most users
/initrd	information contained for booting.. Don't mess with it!
/lib	library files, common files needed by many programs
/lost+found	files that were recovered after a system failure
/mnt	mount point for all external devices
/net	mount point for remote filesystems
/opt	extra third party software
/proc	info about system resources, man proc
/root	home dir. for the the admin
/tmp	temporary space
/usr	program, libraries, docs for most user programs
/var	other temporary files, such as logs, downloaded files, mail queue



Main operations

- cd navigation among directories
- mkdir make directory
- rmdir remove/delete directory
- ls list the content of a directors
- echo $PATH path for executable commands
  eg.: /usr/local/bin/:/usr/bin:/bin/
- export setting environmental variables
  eg. PATH=$PATH:/new/directory/path
- man help for using commands and utilities
- history list of commands used „in the past”
- & after a command: run in background *ls & -> [1] 23142*
- fg bring the job into foreground *fg 23142*

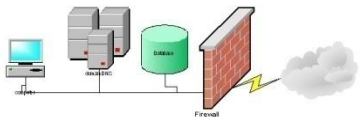


Textual manipulation

- cat catenation of text (printing out textfile content)
- tac last line first (useful e.g. for log files)
- head first lines of a file (default:10)
- tail last lines of a file (default:10)
- more text printout to termial – per page
- less text printout with more user controls

regxp – regular expressions

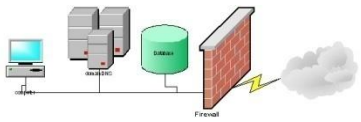
- grep search in text with given rules
- (g)awk text search / manipulation
- sed editing text while running through it once
- vi classical text editor
- emacs cult text editor



Accounting

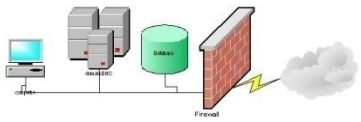
- Multiuser operating system
- Account:
  - Identified by user name,
  - Password protected
- Password file:

  username:password:uid:gid:gecos:homedir:shell
- Account types:
  - Root
  - User



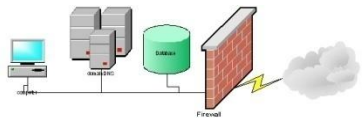
Accounting (cont)

- su — root capabilities change for user
- adduser — adding user
- passwd — changing password
- userdel — deleting user account
- /etc/passwd — user account info
- /etc/shadow — user pasword encrypted / account validity information



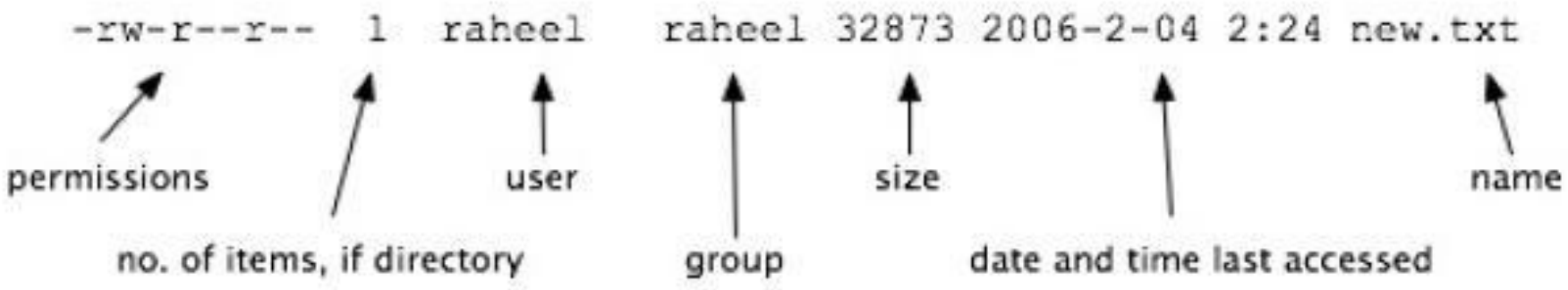
Authorization: ownership & access

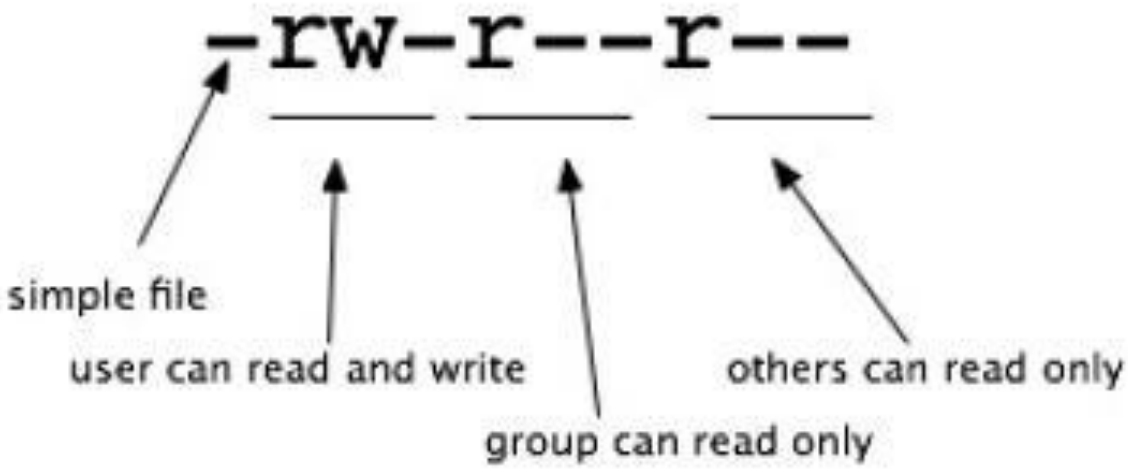
- Access
  - Multi-user environment!
  - Users do not get access to each others' files
  - Special files can only be accessed by root
- Groups
  - Users can belong to many groups
  - Easier / more flexible access control
- Authorization: user / group / others

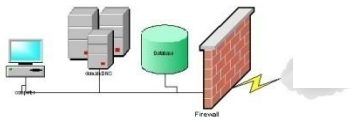


Authorization - example

- ls – l - command output





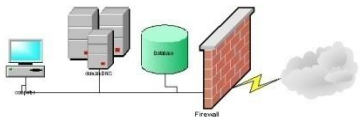


Ownership change

- Ownership change: chown
  chown username file_or_dir

- Group ownership change: chgrp
  chgrp groupname file_or_dir

- Combined group and username:
  chgrp name.name file_or_dir

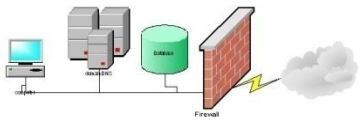


Authorization change

- *chmod*

- *r, w, x: read, write, execute*
- *Root:* can change any user's or file's auth rights
- Beside the *root* only the owner *(user) can change it*

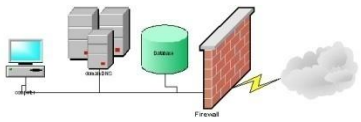
pl. chmod 755 myfile



SCRIPTING IN A (NUT)SHELL

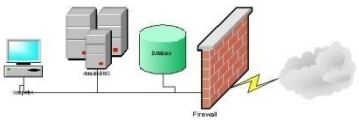
Bash

Suggested tutorial:
http://linuxconfig.org/bash-scripting-tutorial



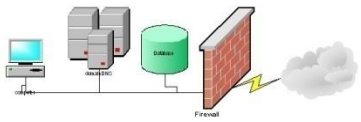
bash - „Bourne-again shell”

- the shell is always available (sh, bash, ksh,..)
- sysadmin scripts: in shell
- more complex script-languages
  - Perl
  - Python
  - Ruby
- Features of script (or interpreter-)languages
  - Source code can be interpreted line-by-line
  - Everything is interpreted run-time
  - string parameter can be interpreted as a command
  - variables handled without types



Process communication channels

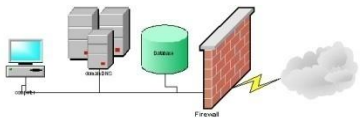
- Standard I/O, for all programs
  - 0 – STDIN
  - 1 – STDOUT
  - 2 – STDERR
- forwarding
  - `tail file` #file→STDOUT
  - `tail file 2>&1` #STDERR→STDOUT
  - `tail file.o > file.n` #file.o→STDOUT→file.n
  - `tail file.o 2> file.n` #file.o→STDOUT, STDERR→file.n
  - grep ‘From’ < /etc/mail/laboruser #grep input: /etc/mail/laboruser



pipes

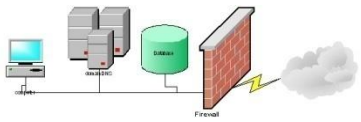
- `cat /etc/mail/laboruser | grep 'From'`
  `#cat STDOUT-ját a grep STDIN-be`
- `cat /etc/passwd | grep ‘:x:’ | sort`
  `#cat STDUT-ját a grep-en átszűrve a sort-ba`

- Piping pros and cons:
  - Unformatted binary data transfer!
  - Fast, but does not handle structured data
  - Structured data must be formatted, then on the receiving side to lines, then fields
  - Utilities for this: cut, awk, sed



bash: variables and names

- No special marking for variabe names
  - etcdir='/etc' # but no spaces around = !! (command)
- $-prefix, if the value of the variable is referenced
  - echo $etcdir
- Can be separated from other text by {} signs:
  - echo ”Saved ${rev}th version of file”



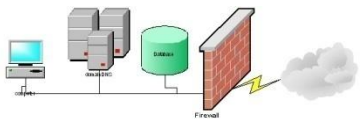
bash: Arrays

```
ARRAYVAR=(Value1 value2 „VALUE3");        #values by listing them
OTHERARRAY[5]= „Value4";        #value to a given index

echo „numitem: ${#ARRAYVAR[@]} ";
echo „first item $ARRAYVAR";

    #for to all items – check where the ; are!

for ((i=0;i<${#ARRAYVAR[@]};i++));
do
   echo "${ARRAYVAR[$i]}";
done;
```

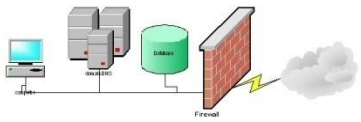



The 'aposthrophies'

(let: mysport=chess)

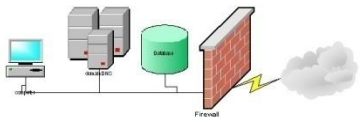
- Double quote is extracting the value:
  - echo ”I play ${mysport}.” # I play chess.
- Simple quote does not extract the variable
  - echo 'I play ${mysport}.' # I play ${mysport}.
- the `backticks`
  - The command in the backticks gets executed, and the output is written out
  - echo ”There are `-wc –l < /etc/passwd` lines in the passwd file.”

    # There are 28 lines in the passwd file.



Special Variables, parameters

- Input parameters
  - `$1, $2` … - input parameters
  - `$#` - number of parameters
  - `$@` - array of parameters
- Previous command
  - `$?` – result value of the last command
  - `$!` – PID of the last background command
  - `$$` - PID of the given shell
- Inter-field separator
  - `$IFS` – based on this, lines are automatically split to fields
  - By default IFS includes all whitespaces `" " "\n" "\t"`



A few common filters

- cut - cuts lines to fields
- sort - sort lines

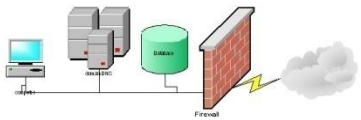
  -f : case sensitive sort

  -k : key column for sorting

  -n : fields are integer numbers

  -r : reverse sort order

  -u: output unique records only
- unique - outputs unique lines
- wc - word count (-l, -w, -c …)
- tee - sends input towards two outputs



bash scripting

((content of helloworld file:))

```
#!/bin/bash
echo ”Hello, world!”
```

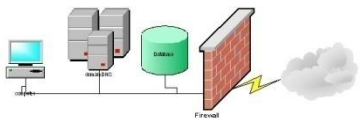
((type in to shell:))

chmod +x helloworld

./helloworld

((shell outputs:))

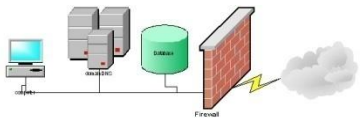
Hello, world!



Conditional statements

```
#!/bin/bash
if [ "foo" = "foo" ];
then
    echo expression evaluated as true
else
    echo expression evaluated as false
fi
```

http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html



Cycles

```
#!/bin/bash
for i in $( ls ); do
    echo item: $i
done
```

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]; do
    echo The counter is $COUNTER
    let COUNTER=COUNTER+1
done
```