PC 2 Corrections

Bash exercices

Write 6 random numbers between 0 and 9 in a file named *randoms* then display them in a sorted way on the screen.

```
echo $((RANDOM % 10)) >> randoms
...
echo $((RANDOM % 10)) >> randoms
cat randoms | sort
```

Or

```
seq 10 | shuffle | head -n 4 >> randoms
cat randoms | sort
```

Bash exercices

Display only the unique numbers of the file on the screen.

cat randoms|sort|uniq

Bash exercices

Add the string : “*This is a string*” at the end of the file. What will return the command *wc {your file}* ?

echo "This is a string" >> randoms
Number of lines, number of words, number of characters
7 10 29

Bash exercices

You have h=”Hello you. Do you enjoyed this course ?”. In one line display : I enjoyed this course.

echo "I ${${h##*Do you }%% ?}."

Bash exercices

Build a shell script to compute the total size of the files in a folder specified as the first argument of the script using a loop to sum the total size

```
#bin/sh

n=0

for file in "$1"/* ; do
  n=$(($(stat --printf="%s\n" "$file") + $n))
done
echo $n
```

Bash exercices

Write a bash script which will print all the file that contains a specific string

```
#bin/sh

echo "$2"
for file in "$1"/* ; do
  if grep --quiet "$2" "$file"; then
    echo "$file"
  fi
done
```

Bash exercices

Write a bash script that compute the result of the operation given in argument

```
#bin/sh

if [ "$2" == "+" ] ; then
  echo $(($1 + $3))
elif [ "$2" == "-" ] ; then
  echo $(($1 - $3))
elif [ "$2" == "*" ]; then
  echo $(($1 * $3))
elif [ "$2" == "/" ]; then
  if [ $3 -eq 0 ]; then
    echo "Error : division by 0"
  else
    echo $(($1 / $3))
  fi
else
  echo "Operation not supported."
fi
```

Google form

cat /proc/cpuinfo | grep ‘name’ | uniq

Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz

cat /proc/version | grep -oE "[0-9.]+" | head -n1

4.9.5

cat /proc/uptime | cut -f1 -d' '

1938.16

cat /proc/stat | grep 'cpu ' | cut -f6 -d' ' && cat /proc/stat | grep 'cpu ' | cut -f3 -d' '

15105.56s idle , 31436 USER_HZ

cat /proc/meminfo | grep 'MemTotal' && cat /proc/meminfo | grep 'MemAvailable'

MemTotal: 16411564 kB, MemAvailable: 9811928 kB

cat /proc/stat | grep ctxt

ctxt 5103534

cat /proc/stat | grep "processes"

processes 9950

Google form

Implement a bash script that computes the average frequency of all CPUs.

```
#!/bin/bash
a=0
n=0
for i in `cat /proc/cpuinfo |grep "cpu MHz"|cut -d \: -f 2|cut -d \. -f 1`
do
a=$a+$i
n=$n+1
done
echo $((a/n))
```

Google form

Update the script so it computes this average at a given interval time (given by the first script argument), N times (second script argument).

```
#!/bin/bash
c=0
l=$2
int=$1
while [ $c -lt $l ]
do
a=0
n=0
for i in `cat /proc/cpuinfo |grep "cpu MHz"|cut -d \: -f 2|cut -d \. -f 1`
do
a=$a+$i
n=$((n+1))
done
echo $((a/n))
c=$((c+1))
sleep $int
done
```

Google form

How should I find more information about the socket system call?

man 2 socket

What does the socket system call return ?

a valid file descriptor or -1 if error

How many arguments has the ioctl system call ?

2+

What is the protocol field to pass to socket for IPv4 ?

AF_INET

What does the function inet_pton do ?

Convert IP addresses to binary

Virtual Machine

Virtual Machine

- Virtualize a computer and all its component
  - Screen → Window
  - Network → Bridge or NAT
  - Keyboard → Keyboard input of the window
  - Hard drive → An image on the disk
  - …
- Multiple level
  - Full → You can install a completely different system, even a different architecture if the CPU is virtualized
    - VirtualBox, QEMU (QEMU/KVM), VMWare, ...
  - OS-assisted
    - XEN, Linux container, …

Why Virtual Machines?

- Why people like VMs?
  - Run Linux inside Windows
  - Snapshots
  - Migration
  - Testing models : Vagrant, docker, ...
  - For kernel/driver development, prevent to crash your computer

Virtual Machine
For kernel testing

We'll test our new kernels with VirtualBox to all be in the (nearly) same environment

The virtual machine is Ubuntu 14.04.3 32bits.

You can download the image at http://queen.run.montefiore.ulg.ac.be/~barbette/cours/info0940/2016-v1.vdi.xz. You **have to** use that one, which is a 32bits ubuntu (remember that all along). With only **one CPU core.**

The user is "student" and the pass "INFO0940".

This machine is only to install, and test your kernel

Windows/Mac Users
Without dual-boot

For **kernel compilation :**

- Install a **second virtual machine** , not using my image but with the Ubuntu **64bit** ISO you can find on ubuntu website, with at least 2 CPU cores !
- **Do not compile a kernel using Cygwin!!!**
- I suggest MAC users to use a second VM.
- I also strongly suggest developing inside Linux too, to avoid “/r” or “/n/r” Mac/Windows problem, hidden Mac OS files, ...

No nested virtualization!

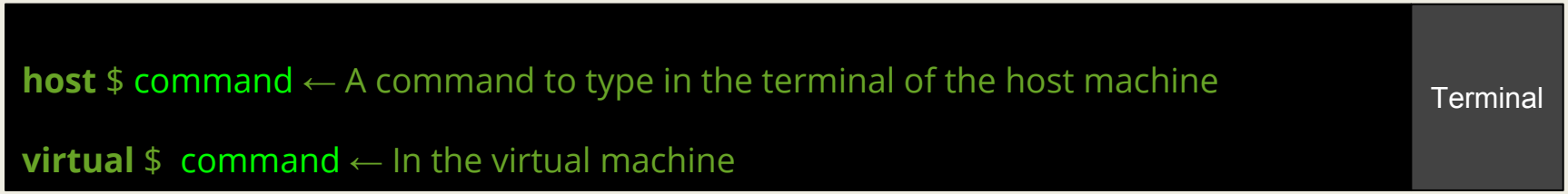
If I hear again something about problems running VirtualBox inside VMWare this year, I will kill you ! (Or even VirtualBox inside VirtualBox…).

Summary
You need two Linux

- One is either a VM or the bare metal OS (dual-boot if you want)
  - For kernel development
  - For kernel compilation
    - Compilation is **very** slow, that's why you should prefer to have a dual boot than a second VM, because VM always introduce performance cost.
- One is VirtualBox VM with the image I provide
  - For kernel testing

Conventions

We'll do things in the **host** machine, and others in the **virtual** machine.



Know where you are ! "sudo make install" on your host will install the kernel in your real machine...

VM Network mode

- **NAT** : VirtualBox act as a normal process which tries to access the web when the VM initiate a connection.
  - \+ Simple
  - \- Virtual machine not accessible from outside, not even from host
- **Bridge** : VirtualBox plugs itself to the real card, and acts as if you used a switch
  - \+ Access from outside (from the world only if you get public IP address)
  - \- Don't work with Wifi (--> do **not** work on **ULG Wifi, but does work on ms8xx machines**)
- **Internal Network** : Network between VMs, invisible to the host or the outside
  - \+ Secure communication between VMs
  - \- No access to or from host or outside (no internet)
- **Host private network** : Like internal network but creates a virtual interface on the host too
  - \+ Fine-tuning is possible (router on the host, …)
  - \- Internet connection tricky to configure (create a router between virtual interface on host and the real interface)

… in practice

- You can create multiple virtual card
- At the ULg with a cable, or at home with a cable, one NIC :
  - Bridge
- On ms8xx, one NIC :
  - Bridge
- On WiFi (ULg or Home), two NICs :
  - NAT (Access to the web from the VM)
  - Host private network (access to the VM from the host)

  → You'll have to use the IP address of the private network to access the host, and as the VM will have two IP adress, you'll have to find the good one… You can do reverse ssh instead of a host private network.

Sharing host files with the VM using sshfs

host $ mkdir /home/jerry/OS

virtual $ mkdir /home/student/OS

virtual $ sshfs jerry@{host ip}:/home/jerry/OS /home/student/OS

sshfs establishes a secure link to a machine where you have an account, and make a sub-tree of this machine's filesystem appear as a sub-tree on the calling machine.

By default, only the user who invoked sshfs can access this sub-tree. I.e. if you invoke it as student on your virtual machine, root may not be able to read content from /remote. You can try the option "**-o allow_other**" or "-o allow_root".

Advantage of sshfs : changes in any side are reflected on the other side

Reverse sharing using sshfs

If, for any reason you cannot address your host, but you can address your virtual machine :

```
host $ ssh -R 2048:localhost:22 student@{virtual ip}
ssh-virtual $ sshfs -p 2048 jerry@localhost:/home/jerry/OS OS
```

Terminal

The first line create a reverse SSH tunnel, so all connection in virtual to port 2048 are transferred to host at port 22.

In NAT mode, virtual is not addressable ! This method won't work !

Copy files from host to VM with rsync

host $ mkdir /home/jerry/OS

virtual $ mkdir /home/student/OS

And one of the following (the second one does not work in NAT mode) :

virtual $ rsync -rapvC jerry@{host ip}:/home/jerry/OS /home/student/OS

host $ rsync -rapvC /home/jerry/OS student@{virtual ip}:/home/student/OS

Copy the file but more intelligently than “scp” : do not copy file which did not changed, use compression, preserve file attributes, ...

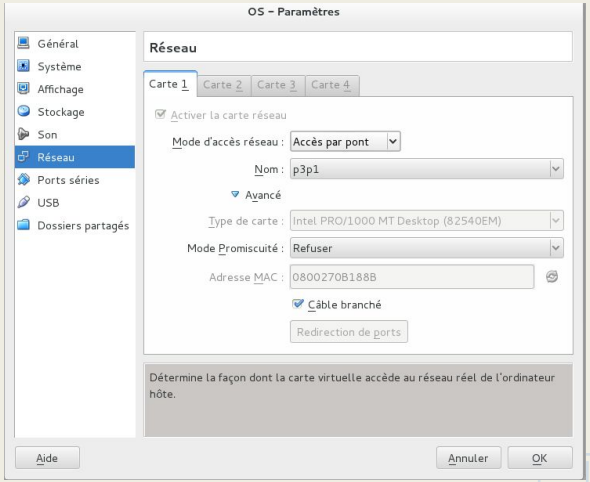
Advantage of rsync : if you mess everything, it was just a copy...

(optional step)

SSH Connection to virtual

I gave you the shortcuts to use the “real” shell, but you may prefer to connect to your virtual machine by SSH.

To access virtual from host, you cannot use NAT for virtualisation of the NIC. Stop your virtual machine, go to network setting, and choose “bridge” mode.



(optional step)

SSH Connexion to virtual (cont'd)

```
virtual $ ip addr     ← Will give you virtual's ip address
host $ ssh student@virtual
```

Terminal

You're now connected to the virtual machine console through ssh, and may benefit from the gnome-terminal rollback, large screen, etc…

Bigger (VM) console

You can change your screen size.

Update /etc/default/grub in the virtual environment and replace

GRUB_CMDLINE_LINUX=""
by
GRUB_CMDLINE_LINUX="vga=791" (for 1024x768, it's 794 for 1280x1024 and 798 for 1600x1200)

```
virtual $ sudo update-grub
```

Terminal

Virtual machine
On ms8xx

- If you can, use your own PC. It's probably faster… And you'll not have space problems
- But it's better than using Windows and playing with multiple VMs…
- **All needed file are already in /virtualbox/REF**
- Account : www.student.montefiore.ulg.ac.be
- You've got public space in /virtualbox for your VM image and so on, just make a folder "g0X" in it. You can use /tmp too (but /tmp will be emptied at each reboot…). You should use it to compile your kernel

Protect your files

- **ls -l** reports *permissions* for owner, group and others as rwxr-xr--
  - on ms8xx, you're all in the same group!
- **chmod go-rwx** *file* restricts all access
  - i.e. permissions become rw------- for a file.
- **getfacl *file*** lists additional control list
- **setfacl -m u:jerry:rw-** would grant the user jerry read/write access to an otherwise restricted resource
  → Use it for other members of your group !

if that's not crystal-clear to you, better check the manpages or
http://computernetworkingnotes.com/managing-file-system-security/acl.html

Introduction to the Linux Kernel

The Linux Kernel

- You should know what is a Kernel by now

- Why Linux ?
  - It’s free
  - It’s open source
  - Widely used
  - Known to be difficult to hack (but better on the CV !)

- A video to cheer you up in bad moments, presenting Linux Kernel :
  - https://www.youtube.com/watch?v=yVpbFMhOAwE

- Its story, and mostly the story of Linus Torvalds
  - https://www.youtube.com/watch?v=XMm0HsmOTFI

The Linux Kernel

- the *kernel image*, ready to be booted:
  - **vmlinuz**, self-extracting compressed image
  - should be installed in **/boot**
- *modules*,
  - mostly hardware drivers, protocols support, ...
  - should be in /lib/modules/<kernel-version>
  - modules are compiled for a *specific* kernel version (even specific build/config)

The Linux Kernel (con't)

- configuration for the *bootloader*.
  - here, the bootloader is GRUB2.
  - auto-generated config by the command “update-grub” goes to /boot/grub/grub.cfg
- a *ramdisk* (initrd)

Compiling the Linux Kernel

Building your own Kernel

Why?

Removing everything you don't want, leading to a very smaller kernel, sometimes faster too. The ubuntu one "have everything" in it, but too much for everyone.

How?

Download and extract the kernel. We'll use the 4.4.50 version (mandatory)

```
host $ mkdir /home/jerry/OS
host $ cd /home/jerry/OS
host $ wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.4.50.tar.xz
host $ tar -Jxvf linux*
host $ cd linux*
```

Configuring the kernel

The configuration of the kernel to build is in the file “.config”

```
#
# Automatically generated file; DO NOT EDIT.
# Linux/x86_64 3.2.35 Kernel Configuration
#
CONFIG_64BIT=y
# CONFIG_X86_32 is not set
CONFIG_X86_64=y
CONFIG_X86=y
CONFIG_INSTRUCTION_DECODER=y
CONFIG_OUTPUT_FORMAT="elf64-x86-64"
CONFIG_ARCH_DEFCONFIG="arch/x86/configs/x86_64_defconfig"
CONFIG_GENERIC_CMOS_UPDATE=y
CONFIG_CLOCKSOURCE_WATCHDOG=y
CONFIG_GENERIC_CLOCKEVENTS=y
CONFIG_ARCH_CLOCKSOURCE_DATA=y
CONFIG_GENERIC_CLOCKEVENTS_BROADCAST=y
CONFIG_LOCKDEP_SUPPORT=y
CONFIG_STACKTRACE_SUPPORT=y
CONFIG_HAVE_LATENCYTOP_SUPPORT=y
CONFIG_MMU=y
CONFIG_ZONE_DMA=y
CONFIG_NEED_DMA_MAP_STATE=y
CONFIG_NEED_SG_DMA_LENGTH=y
CONFIG_GENERIC_ISA_DMA=y
CONFIG_GENERIC_IOMAP=y
CONFIG_GENERIC_BUG=y
CONFIG_GENERIC_BUG_RELATIVE_POINTERS=y
CONFIG_GENERIC_HWEIGHT=y
CONFIG_ARCH_MAY_HAVE_PC_FDC=y
--More--
```

Configuring the kernel

To build a kernel which will work in the virtual machine, you can recover the .config from the Ubuntu kernel running in virtual.

```
virtual $ ls /boot/config*
/boot/config-3.19.0-43-generic-pae
virtual $ cp /boot/config-3.19.0-43-generic-pae ~/OS/linux-4.4.49/.config
```

Terminal

```
host $ make ARCH=i386 oldconfig
```

Terminal

```
host $ export ARCH=i386
host $ make oldconfig
```

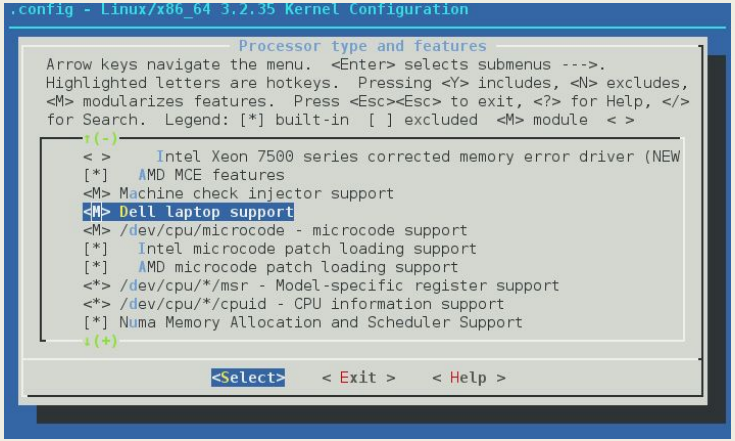
Terminal

Configuring the kernel

And remove what you don’t want from this config (it’s the ubuntu that have to work on all computer in the world… You can imagine it’s not minimalist) with :

```
host $ make ARCH=i386 menuconfig
```

Terminal



Build the kernel

```
host $ make ARCH=i386 bzImage
```
Terminal

→ We set the arch to a 32bit to match our ubuntu version. If you want to build a kernel for your own computer, just type “make bzImage”

```
host $ make ARCH=i386 bzImage -j4
```
Terminal

for 4 threads, -j2 for two, … Use N_Core + 1 for maximized performances. So -j5 for a quad-core CPU.

Then, compile the modules (you’ll need to compile them again only if you modify files chosen to be compiled as modules !)

```
host $ make ARCH=i386 modules
```
Terminal

Install the kernel

Normally you would do :

DO NOT RUN THIS>> host $ sudo make install && sudo make modules_install **<<DO NOT RUN THIS**

When finished, you have to install kernel modules. But if you type the normal “make install && make modules_install”, it will install it in your host !

We’ll install the files in a temporary directory so you can copy them to the virtual machine after that :

Install the kernel (cont'd)

```
host $ mkdir /home/jerry/OS/boot
host $ export INSTALL_PATH=/home/jerry/OS/boot
host $ mkdir  /home/jerry/OS/mods
host $ export INSTALL_MOD_PATH=/home/jerry/OS/mods
host $ make ARCH=i386 modules_install
host $ make ARCH=i386 install
```

Terminal

*The last line produce **errors**, as it will try to install the kernel on the host, it's **normal**, we don't want him to do that, we just want the files in OS/boot*

Install the kernel

(assuming your **sshfs** is still working and with allow_other or allow_root)

virtual $ cd /home/student/OS
virtual $ sudo cp -rf boot/* /boot
virtual $ sudo cp -rf mods/* /

If you haven't allow_root options or have others "sudo" and permission-related problem, first copy files to /tmp as normal user, then copy them from /tmp to their final destination

Finally, we make the initrd image and tell grub (the bootloader) to update itself

virtual $ sudo update-initramfs -u -k 4.4.50
virtual $ sudo update-grub

Did it worked?

virtual $ sudo reboot

Login…

virtual $ uname -a
Linux virtual **4.4.50** #2 SMP (…)

localmodconfig

You can use localmodconfig on the virtual machine to only select current loaded modules and remove all unused module from the .config file. This will speed up the build process a lot. However this sometimes remove too much module.

```
virtual $ make ARCH=i386 localmodconfig
```

Terminal

Use sshfs/rsync to copy your kernel source on the virtual machine and run :

This will update your .config file. Only try this after a first successful compilation and installation, so if there is a problem you know if it’s a missing module or not...

Practical Course 3

PC 3

- Compile the kernel without building all modules (localmodconfig, …)
- Install it in the VM
- Reboot
- Solve bugs

- Build a full local script to re-build the kernel
- Set up auto or easy sshfs in the VM
- Build a full install script in the virtual machine

Unmarked Project 3

CREDITS & Hello !

For **Sunday 18/03 23h59**

Kernel source code

- Download the kernel 4.4.50 source code at https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.4.50.tar.xz
- Extract it inside /home/**$USER**/OS/
- This will create a folder named /home/**$USER**/OS/linux-4.4.50
- Add your names to the **CREDITS** file **at the right place**
- Add a message in the boot process using **printk("Hello INFO0940!");**

Submission

- Make a patch named **gXX-sYY-source.patch** where XX is your group number (assigned on the platform) and YY is the step number (03 in this case)
- Make an empty report but with a good and beautiful first page including your name, your group number, the course title, the step number, …
  - It will be named **gXX-sYY-report.pdf** where XX is your group number
- Pack the two files in an archive named **gXX-sYY-archive.tar.gz**

- Submit the archive using the submission platform

Patch without GIT

1. copy candidate files:
   rsync --exclude "cscope.out" --exclude "scripts/genksyms/*.*.[ch]" --exclude ".tmp_*" --exclude "*.o" --exclude ".git" --exclude "*.log" --exclude "*.ko" --exclude ".mailmap" --exclude ".gitignore" -a linux-4.4.50/ release-4.4.50/
2. cleanup any trailing binary:
   cd release-4.4.50 ; make ARCH=i386 distclean ; cd ..
3. install a clean reference sources nearby
   mkdir clean-4.4.50 ; cd clean-4.4.50 ; tar -Jxf ../linux-4.4.50.tar.xz ; mv linux-4.4.50/* ./ ; rm -rf linux-4.4.50 ; cd ..
4. make the patch:
   diff -burN clean-4.4.50 release-4.4.50 > g42-s01-source.patch
5. delete release-4.4.50 to keep further patchs clean
   rm -rf release-4.4.50

If by the end of the course, you do not have a script to do that automatically, you're doing it wrong…
Keep clean-4.4.50, it won't change !

Patch without GIT (cont'd)

You can test your patch with **patch -p1 --dry-run < ../g42-s01-source.patch** in another clean release of linux-4.4.50

If the path still contains garbage, use the diff arguemnt **-x "file.ext"** to ignore all file named "file.ext", e.g. "diff -x ".gitignore" -burN clean-4.4.50 release-4.4.50"

Remember to check that the patch is clean, by reading it with a text editor. Something you didn't add yourself should not be there. You can also try things "make ARCH=i386 mrproper" or "make ARCH=i386 clean", check the doc...

Patch content

- Make your **patch** as if you would send it to Linus Torvalds himself.
- I will react like him if you sent unclear patch or with things that you didn't changed but were added to the patch for random reasons (generated stuff like includes, or .o files, adding of swap file "file~")
- You may look at his known reaction there to get an idea :
  - One of the best : https://lkml.org/lkml/2012/12/23/75
  - Or, summarized by year :
    - http://flossdata.syr.edu/data/insults/2012LTinsultsLKML.tsv.txt
    - http://flossdata.syr.edu/data/insults/2013LTinsultsLKML.tsv.txt
    - http://flossdata.syr.edu/data/insults/2014LTinsultsLKML.tsv.txt
- But you can leave your comments about discoveries like "//it' call to the driver init function is made"
- Usefull info here : http://kernelnewbies.org/OPWfirstpatch
- If your patch contains binary data → → → → → → →

Questions?

Name : Tom Barbette
Mail : tom.barbette@uliege.be
Office : B37 (Math) 1/13
Site : www.tombarbette.be/courses/os/