



Kernel - Kernel sources

Objective: Learn how to get the kernel sources and patch them.

After this lab, you will be able to

- Get the kernel sources from the official location
- Apply kernel patches

Setup

Go to the `/home/<user>/felabs/sysdev/kernel` directory.

Get the sources

Go to the Linux kernel web site (<http://www.kernel.org/>) and identify the latest stable version.

Just to make sure you know how to do it, check the version of the Linux kernel running on your machine.

We will use `linux-2.6.35`, which this lab was tested with.

To practice the `patch` command later, download the full `2.6.34` sources. Unpack the archive, which creates a `linux-2.6.34` directory.

Apply patches

Install the `patch` command, either through the graphical package manager, or using the following command line:

```
sudo apt-get install patch
```

Download the 2 patch files corresponding to the latest `2.6.35` stable release: a first patch to move from `2.6.34` to `2.6.35` and a second patch to move from `2.6.35` to `2.6.35.x`.

Without uncompressing them (!), apply the 2 patches to the `linux-2.6.34` directory.

View one of the 2 patch files with `vi` or `gvim` (if you prefer a graphic editor), to understand the information carried by such a file. How are described added or removed files?

Rename the `linux-2.6.34` directory to `linux-2.6.35.<n>`.

For your convenience, you may copy the source URL from your web browser and then use `wget` to download the sources from the command line:

```
wget <url>
```

`wget` can continue interrupted downloads

Did you know it? `gvim` can open compressed files on the fly!

Vim supports syntax highlighting for patch files





Kernel - Configuration and compiling

Objective: get familiar with configuring and compiling the kernel

After this lab, you will be able to

- Configure, compile and boot your kernel on a virtual PC.
- Mount and modify a root filesystem image by adding entries to the `/dev/` directory.

Setup

Stay in the `/home/<user>/felabs/sysdev/kernel` directory from the previous lab.

Objectives

The goal of this lab is to configure, build and boot a kernel for a minimalistic, virtual PC, emulated by the `qemu` emulator (<http://qemu.org>)

Kernel configuration

Run `make xconfig` to start the kernel configuration interface. The kernel configuration interface is provided as source code in the `kernel`, `make xconfig` compiles it automatically, but requires libraries and headers. You will need to install the `libqt3-mt-dev` package, which contains the Qt development files, and the `g++` package, the C++ compiler.

In the interface, toggle the `Option -> Show Name` option. This is useful sometimes, when the parameter name is more explicit than its description, or when you're following guidelines which give the parameter name itself.

Also try the `Option -> Show All Options` and `Option -> Show Debug Info` options. They let you see all the parameters which wouldn't appear otherwise, because they depend on the values of other parameters. By clicking on the description of such a parameter, you will see its preconditions and understand why it is not selectable.

Specify a version suffix, so that you can identify your kernel in the running system by running `uname -r` or `cat /proc/version`.

Configure your kernel for a minimalistic PC:

- Pentium-Pro processor (`CONFIG_M686`)
- IDE hard disk (`CONFIG_IDE`, `CONFIG_IDE_GD_ATA`, `CONFIG_IDE_GENERIC`)
- ext2 filesystem (`CONFIG_EXT2_FS`)
- Support for elf binaries (`CONFIG_BINFMT_ELF`)

Take your time to have a look at other available features!

Don't hesitate to ask your trainer for more details about a given option.

Also try with `make menuconfig`. Though it is not graphical, some people prefer this interface. As the `menuconfig` interface is based on the `Ncurses` library, you will have to install the `libncurses-dev` package to use it.

We advise you to unselect all options at once, and add only the ones that you need.



Compile your kernel

Just run:

```
make
```

The qemu PC emulator

qemu is a fast tool which can emulate several processors (x86, ppc, arm, sparc, mips...) or even entire systems.

By using an emulator, you won't have to reboot your workstation over and over again to test your new kernel.

To install `qemu` on your system, simply install the `qemu` package.

Booting your kernel

You are now ready to (try to) boot your new kernel. We will use the `data/linux_i386.img` file as root filesystem.

Back to the main lab directory, run the `run_qemu` script (just adjust the path to the Linux kernel image):

```
qemu -m 32 -kernel linux-<ver>/arch/x86/boot/bzImage \
-append "root=/dev/hda rw" \
-hda data/linux_i386.img
```

If the kernel doesn't manage to boot, and if the error message is explicit enough, try to guess what is missing in your kernel configuration, and rebuild your kernel.

Don't hesitate to show your issue to your trainer.

If you are really stuck, you can try with the rescue config file in the `data/` directory, which your instructor is supposed to have checked.

If everything goes right, you should reach this message:

```
Warning: unable to open an initial console.
```

This happens because no console device file is available in the root filesystem. Without such a file, the shell has no way to interact with the hardware: reading what you type on the keyboard, and displaying the output of commands on the screen.

In our case, the device file the shell is trying to use is `/dev/console`. All you need is to create it!

Type `[Ctrl] C` in the terminal running `qemu` to stop the emulation or close the `qemu` window.

Adding a console device to the root filesystem

Use the following commands in `/home/<user>/felabs/sysdev/kernel` to access the contents of the filesystem image:

```
mkdir fs
sudo mount -o loop data/linux_i386.img fs/
```

Now, create the `dev/console` device that is missing. You can check the `/dev/console` device file on your training workstation to find out the file type as well as the major and minor device numbers.

Here, we won't need to run the `make install` command. If we ran it, the kernel would be installed for the workstation PC, while our plans are to use an emulated PC.

`qemu` is going to emulate a virtual PC with the following features:

- m: specifies its amount of RAM.
- hda: specifies the contents (and size!) of the virtual hard disk.
- kernel: kernel image to boot. `qemu` is here a bootloader too. Before starting the emulation, it copies the kernel file to the RAM of the virtual PC.
- append: options for the kernel. In particular, `root=/dev/hda` instructs the kernel to boot on the first IDE hard disk of the virtual PC.

root permissions are required to create an entry in `/mnt/`, as well as to run the `mount` command

Whatever the architecture Linux runs on, major and minor device numbers are always the same.



Once this is done, unmount the root filesystem:

```
sudo umount fs
```

Rerun your `qemu` command. Now, you should reach a command line shell.

Run a few commands in the virtual PC shell.

Kernel version

Query the version of the running kernel and make sure you find the version suffix that you specified at configuration time.

Conclusion

Well done! Now you know how to configure, compile and boot a kernel on a minimalistic PC.

Going further

If you completed your lab before the others...

Add framebuffer console support to your kernel, and choose the VGA 16 color framebuffer driver. Then, add boot logo support. You should now see a penguin logo when your virtual PC boots.

Add SMP (Symmetric Multi Processing) support to your kernel. Using the `-smp` option of `qemu`, simulate a PC with 4 processors. Because of a temporary `qemu` bug you will also have to add the `noapic` parameter to the kernel command line (passed by the `-append` parameter of `qemu`), You should now see 4 penguins on the framebuffer console, indicating the number of CPUs found on your system!

Add framebuffer console rotation support to your kernel, and modify the kernel command line (`-append` parameter again) to boot your kernel with a 90 degree anticlockwise rotation.

If you don't unmount a filesystem or do not use special mount options, you can't be sure that your changes have already been committed on the physical media (the `.img` file in this case).

To unmount a filesystem, remember that you must be *outside* of the directory where the filesystem is mounted. Otherwise `umount` will fail with `Device or resource busy`.

Don't be surprised if the whole kernel sources are recompiled after enabling SMP support. This changes many data structures throughout the kernel sources.

You will find details about how to use console rotation in the [Documentation/fb/fbcon.txt](#) file.

