Tutorial / Lab n° 1
Linux Kernel and System Calls

The purpose of this lab is to familiarize yourself with the Linux kernel. You will start by discovering virtualization and start to use a virtual Linux system upon you workstation. You will continue by configuring and compiling a kernel before adding new features to discover what system calls are and how to define them.

1 Installing a development environment: VMware & Qemu

To compile a new kernel, we use a Linux machine with all the necessary compilation tools. To stay in our favorite environment and to facilitate recovery in case of kernel panic that you will surely have, we will work with a virtual machine.

1.1 Installing VMware

First of all, you will install a virtualization environment with the help of VMware to be able to have an efficient compiling environment. You just have to download and install the following package and follow the screen instructions

`http://mymachine/esa/lab1/install/VMware-player-3.0.1-227600.exe`

1.2 Installing Qemu

We will use Qemu in a second time to be able to emulate embedded systems as qemu is an emulator. We will configure it now be we will use it later.

1.2.1 Host machine configuration

To be able to enjoy network on your guest machine, you must first install openvpn which will result to create a tap interface.

`http://mymachine/esa/lab1/install/openvpn-2.0.9-install.exe`

1.2.2 Guest machine installation

To implement a complete emulator system, we'll use qemu. To do this, retrieve the package qemu for your host system and install them on your machine.

`http://mymachine/esa/lab1/install/`

1.2.3 Configure your virtual machine and create a disk image

You have to create a disk image for your virtual machine with at least 8GB (16GB offers more flexibility knowing that qcow2 is a format that allow the image file to grow when required).

Configure your virtual machine for x86 32-bit microprocessor, 256MB of RAM, a PCI NE2000 network card. Be sure to configure your host machine and your virtual machine to access the Internet from your virtual machine (using tap interface created above and sharing your Internet access on your host machine as described during the lecture. Beware of the used interface).

2 Installing a GNU / Linux operating system

2.1 Installing a system from a virtual machine disk image

To obtain a system already configured to work with, you can recover the disk image of the virtual machine to the following address:

`http://mymachine/esa/lab1/hda-system.qcow2.bz2`
`http://mymachine/esa/lab1/hdb-linux-kernel.qcow2.bz2`

Add to your VMware setup these two virtual disks to the virtual machine you have configured. You should now be able to start your virtual machine.
You can update your Debian GNU / Linux with the help of the following commands

```
apt-get update  (met à jour la liste des paquetages disponibles)
apt-get upgrade  (installe les mises à jour nécessaires)
```

2.2 Installing a complete system from scratch

If you do not have disk image containing the system already installed and to have a system that allows us to make the kernel compile on Linux, get the disk image to install Debian GNU / Linux 5.03 on i386

```
http://mymachine/esa/lab1/debian-503-i386-businesscard.iso
```

You will create IDs for root (root password) and for a standard user: user (user password). You are able to install from the network that you just configured.
In Debian, if you are missing a package, you can get it through `apt-get install` if you know the name. If you don't, you can use `aptitude` or `dselect` to select the desired package in the list of available packages. Be sure to bring your system up to date and install all necessary packages for the lab, especially for a complete toolchain (gcc, make, ...).

3 Getting Started with Linux kernel

3.1 Installing a new Linux kernel

3.1.1 Installing kernel sources from an existing disk image

If you have not already done, get at a disk image containing the kernel sources for the lab:

```
http://mymachine/esa/lab1/hdb-linux-kernel.qcow2.bz2
```

Mount image in the tree of your file system in `/work` :

```
mount /dev/hdb1 /work
```

You can now access th linux kernel source code through `/work`.

3.1.2 Installing kernel sources from a source package

Get the kernel sources at the following url (as well as the signature file to verify the authenticity of the package):

```
wget http://mymachine/esa/lab1/linux-2.6.30.10.tar.bz2
wget http://mymachine/esa/lab1/linux-2.6.30.10.tar.bz2.sign
```

You just have to check the archive with the given key `kernel.org` with the identifier `0x517D0F0E`:

```
gpg --keyserver wwwkeys.pgp.net --recv-keys 0x517D0F0E
gpg --verify linux-2.6.26.3.tar.bz2.sign linux-2.6.26.3.tar.bz2
```

and to install kernel sources in directory `/usr/src`.

3.2 Getting familiar with the kernel sources

As a new Linux kernel developer, you have to find out where a particular feature is in the sources. You will have to navigate through out more than 320MB of sources and literature, more than 8,000,000 lines of C code ... It is therefore important to be "familiar" with the organization of the kernel sources!

You can help yourself with the standards Unix tools as `locate`, `find`, `grep`, `ctags` ... but you can also use the Linux Cross Reference website : http://lxr.linux.no/

First, you can look for the following information:

- The source code if the ne2k driver (the network card emulated by the qemu emulator),
- The documentation about DVB-T (Digital Video Broadcast) supported by your version of kernel,
- ...

4 Configuring and Compiling Kernel

4.1 Configuring kernel

We want to build a Linux kernel configuration with limited resources (as we will work on embedded systems). We must therefore include in this configuration the minimum drivers to minimize memory used by the kernel (by the way, it will gain valuable time compilation also).
You should generate a minimal kernel version (`make allnoconfig`) and only add the needed drivers (as module or building, depending on the machine configuration you created). As we are under terminal Console, to configure the kernel you use `make menuconfig` and before we need to install the package `libncurses-dev` if it's not already done.

4.2 Installing and testing the created kernel

4.2.1 Installing kernel image on your system

At first, install the kernel you compile on the system your used for compiling. Change it to the kernel loader that is installed on your system (grub) to offer this new kernel at next reboot, but be sure not to use this new kernel as the default one.

4.2.2 Using the kernel with another system

In this second step, you will use the kernel you compiled to start a new virtual machine. For this, you need to mount a shared folder on your host machine on the filesystem of the guest machine (GNU / Linux). This step will allow you to get the compiled files outside your virtual machine.
To start the newly compiled kernel with another virtual machine (a qemu one), remember to use options `-kernel bzImage -append cmd -initrd file` de `qemu`.

5 System Calls

5.1 Adding a new system call to the kernel

To understand how a system call works, we will simply create new one. Taking the example model given during lecture, add the following system calls to the Linux kernel:

```
int addition(int x, int y, int *val) ;
void increment(int *val) ;
```

Add these two system calls to the kernel. To achieve that first point, add a look to slides of lecture (from 71 to 74).
Once the new system call added to the kernel, you have to recompile your kernel, install it on your system, reboot your system, taking care of booting the kernel you just have made.
Then you have to write and compile a small C program to be able to test the new functionalities you added to your kernel. You would take care try impossible addresses (or write-protected) for the `val` parameter.