# Tutorial / Lab n° 3
# Free Software for Embedded System

The purpose of this lab is to familiarize yourself with some free software in embedded systems.

## 1 Configuring kernel for embedded system

First, you should verify that the following options are enabled in the kernel configuration file:

- Ext2 should be natively supported
- BLK_DEV_RAM should be enabled

## 2 Creating a File system for embedded system

### 2.1 Creating a root file system

Create a file of 2MB which will allow us to store the image of our file system (called `root.img`). Make the formatting of this virtual disk in `ext2`. You take care to mount the file created in your main tree to facilitate data exchange.

```
dd if=/dev/zero of=root.img bs=1024 count=2048
mke2fs -m 0 -i 1024 -F root.img
mount -o loop root.img /mnt/img
```

### 2.2 Installing BusyBox

Download the current version of BusyBox:

```
wget http://kistren.polytech.unice.fr/cours/iam03/td5/busybox-1.11.2.tar.bz2
```

Start by creating a light/compact configuration (reminder: you can use `make allnoconfig` to achieve that). Then, configure BusyBox to include the minimum tools necessary to our file system. We will need to perform the following actions on the target. So you should enable options to be able to do the following actions:

- Create, delete and display directories, files and special files
- Allow the cleaning of the console, change the keyboard (`dumpkmap loadkmap`)
- Create a file with vi editor
- Start init process at boot
- Allow stopping and restarting the machine
- Mount partitions
- Setting the target network (`ifconfig`, `route`, `ping`)
- Start a Web server on the target (without authentication, but with support cgi)
- Stop a process
- Use the uptime command to find out how long the system has started
- Have a shell (`ash`)

Remember to create an executable statically compiled. You will also ensure configure BusyBox not use `/usr`.
Compile BusyBox for x86 target.
Copy all files obtained (`_install`) in your file system. Verify that all controls are links to the executable compiled statically.

### 2.3 First system configurations

#### 2.3.1 Add the minimum files to /dev manually

Add to your root file system `/dev/console` in order to interact with your system via a shell.

```
mknod dev/console c 5 1
```

You can also create entries `tty` and `tty[0-5]` to ensure that the system does not send error messages when booting.

#### 2.3.2 Configure the keyboard on the embedded system

To configure the keyboard, you should use `loadkmap` on your target machine. It must pass a file on standard input for the set.

```
loadkmap < /etc/mykbd.kmap
```

To create this file from your machine, run the following commands:

```
busybox dumpkmap > /tmp/mykbd.kmap
cp /tmp/mykbd.kmap [to /etc on the embedded system]
```

You can make your changes permanent by issuing the sync command that flushes all buffers in writing by making entries on the file system of the target.

### 2.4 Start the system with the new created file system

Be careful how you make entries on the file system. It is not recommended to have two systems accessing the same file system (both host and target machine).

Test launch of your system in a virtual machine (`qemu`).

```
qemu –L . –m 32 –kernel vmlinuz-2.6.26-8 –hda root.img –append "root=/dev/hda"
```

At this stage, you should go to a console and be able to run commands that you compiled with BusyBox.

## 3 Advanced configuration

### 3.1 Root file system of your embedded system

Create the `/proc` entry on the root filesystem of your target.

Mount the virtual file system `/proc`. Now that `/proc` is available, you may find that you can turn your actual target using the `halt` command. This will enable you to make sure that all changes to the file system have been saved.

### 3.2 System configuration for starting system

Create the `/etc/inittab` file. You can find documentation on the syntax supported by BusyBox on the website of BusyBox.

```
# Ceci est un script pour init
::sysinit:/etc/init.d/rcS
# Démarrer un shell "askfirst" sur la console
::askfirst:-/bin/sh
# Choses à refaire au redémarrage d'init
::restart:/sbin/init
# Choses à refaire au rédémarrage de la machine
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount –a -r
```

Create the script `/etc/init.d/rcS` defined in the file `/etc/inittab`. In this configuration script, you will see to automatically load the correct configuration of your keyboard and mount `/proc` file system.

# 4 A Web server in your embedded system

## 4.1 Network configuration

Add the options `-net nic,model=ne2k_pci` and `-net socket,listen=localhost:1234` to add a new interface to your working station. You will configure the new network interface `eth1` with address `10.0.2.15`.

Add the options `-net nic,model=ne2k_pci` and `-net socket,connect=localhost:1234` to the command line of `qemu` to add a new network interface to your target system. You will configure this new interface with address `10.0.2.16`.

With that you just have configured a VLAN between your two virtual machines.

Test the connection from the host machine to target machine and vice versa using the `ping` command. You will watch over the target machine using the `-c` option to specify the number of ping made before the stop order. It is indeed likely that the `Ctrl-C` does not work; this problem is due to a misconfiguration of your terminal.

## 4.2 Configure the web server of your target

Create directories `/www` and `/www/cgi-bin` on the file system of your target system. You would add the following cgi script to the directory /www/cgi-bin/ :

```
#!/bin/sh
echo "Content-type: text/html"
echo ""
echo "<html><header></header><body>"
echo "<h1>Durée de fonctionnement:</h1>"
echo "Votre système embarqué tourne depuis: <pre>"
echo `uptime`
echo "</pre></body></html>"
```

Start the BysyBox http web server with the following command line:

```
/sbin/httpd –h /www/ &
```

Test that you can access the web server of your target, and that's done!

# 5 Conclusion

Using the parameter `-m` of `qemu` (to specify how much memory the virtual machine), try to find what is the minimum amount of memory you can use your actual target system (a binary search will give you the results as quickly of course!)