



## Sysdev - Cross-compiling BusyBox

Objective: Learn how to configure, build, test and deploy BusyBox for your target platform.

After this lab, you will be

- More familiar with the BusyBox capabilities and options
- Able to configure BusyBox options
- Able to configure BusyBox in a cross-compiling environment
- Compile and test BusyBox.

### Setup

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

Also type the below command:

```
sudo sh -c "echo 4096 > /proc/sys/vm/mmap_min_addr"
```

### Getting and configuring BusyBox

Get the latest stable version of BusyBox from <http://busybox.net/>.  
Unfold the source archive and enter the source directory.

Now, configure BusyBox to choose what features it will include.

```
make xconfig
```

Take your time to read all the options. This way, you will see all the capabilities of BusyBox.

In order to see how big the BusyBox executable can be, choose to build a static executable (no shared libraries). Just choose the features needed in starting and using a basic system. You can accept the default choices if you wish.

Modify the Makefile to cross-compile BusyBox for arm, using the cross-compiler you've just built:

```
/usr/local/xtools/arm-unknown-linux-uclibcgnueabi/bin/arm-linux-gcc
```

Now cross-compile BusyBox:

```
make  
make install
```

If a failure happens when compiling BusyBox, it is very likely to be because of a capability missing in the toolchain. If this happens, go back to the configuration interface, and disable the feature you didn't manage to compile.

Look at the contents of the `_install/` directory. It's a complete root filesystem structure which can be used in an `initrd / initramfs` or in a small embedded system.

### Testing BusyBox

With the `qemu` emulator, we can even test BusyBox even before installing it on a target machine!

This setting is required to execute ARM executables with the `qemu-arm` command at the end of the lab. We will make this setting permanent in the "System development with Scratchbox" lab. This lab was last tested with BusyBox 1.9.0.

Note the BusyBox uses the same configuration tools as the Linux kernel: configuration settings stored in a `.config` file, same syntax, same interfaces.

See how all the executables are links to the single BusyBox binary. When it is called, BusyBox checks the command name if was given, such as `ls`, `cat`, or `gzip`.



It's easy. Just run:

```
qemu-arm ./busybox ls
```

If everything works, this means that the toolchain you built works fine (in case you used your own), as well as your BusyBox build.

Here we are using qemu's user emulation, which just emulates the target processor, not the whole system. It can run userland binaries, but when a system call is made, it is the native system function (here i386 Linux) that is called.