Java for Linux embedded systems
Training lab book

Thomas Petazzoni, Michael Opdenacker
Free Electrons
http://free-electrons.com

About this document

This document is part of an embedded Linux training from Free Electrons.

You will find the whole training materials (slides and lab book) on http://free-electrons.com/training

Lab data can be found on http://free-electrons.com/labs/embedded_linux.tar.bz2.

Copying this document

Document updates and translations available on http://free-electrons.com/docs/java.

Corrections, suggestions, contributions and translations are welcome!

Training setup

See the training labs on http://free-electrons.com/docs/kernel for setup instructions, which are shared with these practical labs.

Java lab

Objective: Cross-compile and install a Java class library and virtual machine

After this lab, you will be able to

- Cross-compile and install the GNU Classpath class library
- Cross-compile and install the JamVM virtual machine, a lightweight VM for embedded systems
- Compile and run simple text-only Java applications, on an ARM target.

Environment setup

In `/mnt/labs/java/lab1/`, you will find:

- A `data/nfsroot/` directory, that contains a root filesystem that you will use as a base for this lab. It contains a Busybox installation, the basic libraries associated to the cross-compiling toolchain, and the Zlib library, required by JamVM.
- A `data/zImage` file, which is an ARM kernel for the Realview platform emulated by Qemu. The kernel supports booting with the root filesystem over NFS.
- A `run_qemu` script that allows to run the Qemu emulator with the correct arguments

Make sure that your NFS server exports `/mnt/labs/java/lab1/nfsroot/`, and try to run Qemu with the `run_qemu` script. You should access a shell prompt, thanks to Busybox.

To compile the class library and the virtual machine, you will need a cross-compiling toolchain that matches the given root filesystem. To install such a toolchain, add the following line to your `/etc/apt/sources.list` file:

```
deb http://www.free-electrons.com/labs/ubuntu/ ./
```

Then, run `apt-get update` and `apt-get install buildroot-uclibc-arm-toolchain`

The toolchain is now installed in `/usr/local/uclibc-0.9.28-2/arm/`. Add the `/usr/local/uclibc-0.9.28-2/arm/bin/` directory to your path, so that the `arm-linux-*` tools can be easily used for cross-compiling.

Class library

The class library we will use is GNU Classpath, available from http://www.gnu.org/software/classpath/. Download version 0.97.2, that we tested for this lab, and uncompress the tarball.

Most of class library is implemented in Java, so we need to install a Java compiler. Such a compiler is available in the `java-gcj-compat-dev` in Debian and Ubuntu distributions. We don't need a Java cross-compiler because Java bytecode is portable!

Now, configuring GNU Classpath is done using the classical configure script:

```
./configure --host=arm-linux --prefix=/usr --disable-gtk-
peer --disable-gconf-peer –disable-plugin
```

We disable the GTK peer so that the support for graphical applications is not compiled in, and the set of dependencies is reduced. We also remove the dependency on Gconf, and disable the compilation of the Web plugin.

Then, compile by simply running `make`

After the compilation process, you can install GNU Classpath to the target root filesystem using:

```
make DESTDIR=/mnt/labs/java/lab1/data/nfsroot/ install
```

Take a look at the files installed by GNU Classpath. There are some native cross-compiled libraries in `/usr/lib/classpath/`, but the most important part is the `/usr/share/classpath/glibj.zip` file, which contains the implementation of many standard Java classes.

Compiling the virtual machine

Download JamVM 1.5.1 from its official website, http://jamvm.sourceforge.net/. Uncompress the tarball and configure JamVM as follows:

```
CPPFLAGS=-I/mnt/labs/java/lab1/data/nfsroot/usr/include/
LDFLAGS=-L/mnt/labs/java/lab1/data/nfsroot/usr/lib/
./configure --host=arm-linux --prefix=/usr –with-
classpath-install-dir=/usr
```

We need to set `CPPFLAGS` and `LDFLAGS` to allow the configure script to find the Zlib library, used by the Java virtual machine to open the `.zip` files which contains the standard Java classes of the class library.

After the configuration step, compile JamVM by running `make`, and install it using:

```
make DESTDIR=/mnt/labs/java/lab1/data/nfsroot/ install
```

Testing our Java installation

In `/mnt/labs/java/lab1/data/nfsroot/root/`, create a new `Test.java` file that contains:

```
class Test {
      public static void main(String args[]) {
            System.out.println("This is a test");
      }
}
```

Then, compile this Java application using `ecj`:

```
      ecj Test.java
```

This compilation will generate the `Test.class` file, that contains the `Test` class, compiled as bytecode. Now, run your embedded system, and inside the `/root/` directory, run:

```
        jamvm -Xmx2M Test
```

And see your Java application running !

Note: the `-Xmx` argument also to reduce the maximum size of the heap to 2 megabytes. By default, the maximum size is 16 megabytes, and because our emulated embedded system only has 16 megabytes of memory, we need to reduce the size of the Java heap.