Real Time Operating Systems

Building a Test Filesystem

Luca Abeni

Testing CrossCompiled Binaries

- We know:
  - How to compile an ARM executable
  - How to execute it by using Qemu
    - Remember: for dynamic executables, we need the `-L` option to tell qemu where shared libraries are
- But how to build a bootable filesystem for an embedded device?
- We need at least some basic directories:
  - `/etc`: contains some boot scripts and configuration files
  - `/lib`: needed if we use dynamic executables. Contains shared objects
  - `/bin`, `/sbin`: contain the executable files

Filesystems in Embedded Devices

- First problem: how to generate executables for all the commands in `/bin` and `/sbin`?
- Second problem: embedded devices generally do not have hard disks...
  - The filesystem is saved in flash disks... Smaller than regular disks
  - The filesystem is mostly read-only...
- Solutions:
  - BusyBox (`www.busybox.net`) implements all the commands we need (and more!)
  - Use a ram FS. Not a real filesystem... Only a collection of files read by linux on boot and saved in a “fake” filesystem.

Compiling BusyBox

1. Download `busybox-1.3.2.tar.gz` and untar it
2. Be sure that the compiler is in your path (`export PATH=...`)
3. `make ARCH=arm CROSS_COMPILE=arm-unknown-linux-gnu- menuconfig`
   - Enable `mount`, `mkdir`, `mdev`, `ls`, `echo`, `ash`...
4. `make ARCH=arm CROSS_COMPILE=arm-unknown-linux-gnu-`
5. `make ARCH=arm CROSS_COMPILE=arm-unknown-linux-gnu- install`

Simple etc Scripts

- Write this in `_install/etc/init.d/rcS`:

```
#! /bin/ash

mkdir -p /proc
mount -t proc proc /proc
mkdir -p /sys
mount -t sysfs sysfs /sys
mkdir -p /dev/pts
mount -t devpts devpts /dev/pts
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s
hostname TEST
ifconfig lo 127.0.0.1 up
/bin/ash
```

- Write `_install/etc/passwd`:
  `root::0:0:root:/root:/bin/ash`

Last Steps

- Copy the dynamic libraries in the target fs: `cp -a .../arm-unknown-linux-gnu/lib _install`
- `ln -s /etc/init.d/rcS _install/init`
- `cd _install`
- `find .  | cpio -o -H newc | gzip > ../ramfs.img`
- To test with qemu, we need an ARM kernel...
- Get the `http://www.qemu.org/arm-test-0.2.tar.gz` package from the qemu web site

Testing the Image

- Unpack `arm-test-0.2.tar.gz` somewhere: `tar xvzf arm-test-0.2.tar.gz`
- Run `qemu-system-arm` with the kernel from `arm-test-0.2.tar.gz`:

```
qemu-system-arm -kernel arm-test/zImage.integrator -initrd
.../busybox-1.3.2/ramfs.img
```

- Note: `"-initrd <your image>"`
- You can use `-nographic -append "console=ttyAMA0"` to run in text mode
- Exercize: can you repeat everything for `x86`?

Obtaining a Kernel

- We got `zImage.integrator` from a precompiled package
- How to compile it?
  - Need to compile the linux kernel from sources
  - ARM target $\rightarrow$ cross-compilation is needed
  - It is very important to properly compile the kernel
- A big amount of disk space is needed $\rightarrow$ not possible with a 100MB quota
- Preliminary steps:
  - Download the linux kernel source from `http://www.kernel.org`: linux-2.6.x.tar.bz2
  - Uncompress the tarball: `tar xvjf linux-2.6.x.tar.bz2`

Compiling the Linux Kernel

- `cd .../linux-2.6.x`
- Download the kernel configuration file from `www.dit.unitn.it/~abeni/RTOS/arm-linux-config` and copy it in `.config`
- `make ARCH=arm CROSS_COMPILE=arm-unknown-linux-gnu- oldconfig`
- `make ARCH=arm CROSS_COMPILE=arm-unknown-linux-gnu-`
- The compiled kernel is now in `arch/arm/boot/zImage`