Instituto Superior de Engenharia do Porto
Mestrado em Engenharia Eletrotécnica e de Computadores
Arquitetura de Computadores

Cross-compilation with Buildroot

Introduction

Buildroot is a tool that can be employed to generate embedded Linux systems, directly from the source code, through cross-compilation. In this tutorial, we will use Buildroot to build a minimalist distribution targeted to the Raspberry Pi system.

The Raspberry Pi system is based on a processor with ARM architecture. Therefore, in order to create the distribution, we will need at least a compiler that can generate code for that architecture. In this case, we will create a toolchain specially targeted for the Raspberry Pi system. Typically, besides the compiler(s), assembler and linker, a toolchain contains a C library and may contain debugging tools. There are publicly available pre-built toolchains for ARM processors (see, e.g., https://elinux.org/Toolchains). Buildroot allows us to create a toolchain from the scratch, and, in the process, to choose the most suitable versions of the compiler and C library, among several other configuration parameters.

Initial setup

Buildroot requires several packages to be installed on the host system (your workstation). Start by executing the following commands:

```
su
dnf install binutils bzip2 unzip gcc make autoconf automake libtool
dnf install patch git subversion bison flex gettext gawk texinfo gperf
dnf install quilt screen
dnf install qt-devel ncurses-devel perl perl-ExtUtils* zlib-devel gcc-c++
exit
```

Some of these packages might be already installed, and additional packages may also be required. If additional packages are required, Buildroot fails during the compilation giving some hint regarding the missing package.

Using buildroot to create a toolchain and a minimal root file system

1) Download the most recent stable version of the Buildroot package from https://buildroot.org/download.html, extract the archive contents to you working directory and rename the Buildroot directory:

```
cd ~
mkdir -p student_number/Lab3
cd student_number/Lab3
tar xvf ~/Downloads/buildroot-VERSION.tar.bz2
mv buildroot-VERSION buildroot-rpi
```

You should be careful when choosing the working directory, because the generated toolchain will be bound to the chosen directory.

2) Buildroot has several predefined configurations for different target systems (the complete list can be obtained with the command **make list-defconfigs**). In this experiment, we will use the default configuration for Raspberry Pi. The generated system will run on Raspberry Pi and later systems, and also on the QEMU emulator system, with some limitations.

```
cd buildroot-rpi
make raspberrypi_defconfig
```

3) Launch the configuration screen and inspect the default configuration:

```
make xconfig
```

Under "`Target Options`", make sure that the "`Target architecture`" is set to "`ARM (little endian)`", with "`Target Architecture Variant`" set to "`arm1176jzf-s`".

Under "`Toolchain`":

- For "`Kernel Headers`", choose "`Manually specified Linux version`". Then, click "`linux version`" and type **4.14.50**

  The kernel headers provide the interfaces to be used by the system libraries to access the kernel services. The kernel headers version must be the same or older than the kernel that is going to be used in the final system. Although version 4.14.50 is not the latest available kernel version, it corresponds to a publicly available kernel specially configured to run on QEMU, which may be useful at a later time.
- Set "`Custom kernel header series`" to **4.14.x**
- Make sure `uClibc-ng` is selected as the "`C library`";
- Check "`Enable toolchain locale/i18n support`";
- Keep the default choices for "`Binutils Version`" and "`Gcc compiler version`".
- Check "`Enable C++ support`"

Under "`System configuration`":

- Change the "`System Banner`" to your student number;
- Change the "`Passwords encoding`" to "`sha-512`";
- Use "`busybox`" as "`init system`";
- Use "`Dynamic using devtmpfs only`" for "`/dev management`";
- "`Enable root login with password`" and set the `root` password to "`root`". Note that some services, such as `sshd`, require the `root` account to have a password.

Under "`Kernel`":

- Uncheck "`Linux Kernel`".

Under "`Filesystem images`":

- Check "`ext2/3/4 root filesystem`" and choose the `ext4` variant.

4.1) Save your configuration (`File->Save`) and leave the configuration screen.

4.2) In order to avoid errors due to the fact that no kernel will be generated, execute the following commands to generate empty files with the names of the expected files:

```
mkdir -p output/images
touch output/images/bcm2708-rpi-b.dtb
touch output/images/bcm2708-rpi-b-plus.dtb
touch output/images/bcm2708-rpi-cm.dtb
touch output/images/zImage
```

Note that the generated `vfat.img` and `sdcard.img` files will not be valid.

5.2) The system is ready to be built. To proceed, just type

```
make
```

The building step may be quite long, involving several downloads. This will build the *toolchain* and the root file system for the target system. It is possible to interrupt the process, by pressing `ctrl+c`, and to resume it later by executing the `make` command again.

Open a new terminal to continue the tutorial, and let the compilation proceed in the initial terminal.

Preparation of the boot storage device

6.1) The Raspberry Pi processor expects to find a VFAT file system at the offset 2^22 of the boot device. This file system must contain the bootloader and kernel files (firmware). We will use the firmware provided by the Raspberry Pi developers. The image of the VFAT partition is available at the course website in compressed format (gunzip):

```
cd ~/student_number/Lab3
wget http://ave.dee.isep.ipp.pt/~jes/arcom/Lab3-Buildroot/vfat.img.gz
gunzip vfat.img.gz
```

Additionally, the Linux kernel expects to find a Linux file system (EXT2/3/4) containing the root file system.

In what follows, we will create the two partitions that will hold each file system: one for the boot file system (VFAT) and another for the root file system (EXT4).

6.2) Plug your USB storage device and identify the respective device file (e.g., `/dev/sdb`) by inspecting the size of each detected device in the list:

```
lsblk
```

The command above displays a tree representation of all storage devices and respective partitions. It also displays the mount point for each mounted file system.

Alternatively, if the `lsblk` utility is not installed in your Linux distribution, you can use the following command:

```
cat /proc/partitions
```

Make sure that all partitions of your USB device are unmounted. For example, assuming the device is associated with `/dev/sdb`:

```
su
umount /dev/sdb*
mount | grep sdb
```

Repeat until done: note that a partition can have been mounted more than once; therefore, you should check the system state with the mount command, and repeat the above procedure if necessary, until you get a message such as:

```
umount: /dev/sdb: not mounted.
umount: /dev/sdb1: not mounted.
```

6.3) In what follows, we will erase the partition table of the USB storage device. Moreover, we'll try to erase the file allocation tables of any previously existing partition.

Note that you will lose the contents of your storage device. You should only do this if you have a backup of the data or if the data are unimportant.

This is done by setting the first 20 MB of the device to zeros:

```
dd if=/dev/zero of=/dev/sdb bs=1M count=20
```

6.4) Create two primary partitions on the device, using the `fdisk` command.

```
fdisk /dev/sdb
```

For the first partition:

- Enter 'n' followed by the *Enter* key
- Press enter to create a primary partition
- Press enter to create it as partition 1
- For the the first sector, enter **8192** (corresponding to 8192 sector x 512 bytes/sector = 4 MiB).
- In order to define the last sector of the partition, enter the size of `vfat.img` in sectors (minus one), i.e., **+88471** (45297664/512 = **88472** sectors). DO NOT forget the "+" sign.
- **Enter 't' to change the partition type. The first partition type should be set to type 'c' (W95 FAT32 (LBA)).**
- Press 'a' to mark partition 1 as boot partition.

For the second partition:

- Enter the ‘n’ command again
- Press enter to create a primary partition
- Press enter to create it as partition 2
- For the first sector, enter **98304**, the same value used by the Raspbian distribution (the minimum requirement is to be greater than **8192** + **88472** = **96664**)
- For the last sector, press enter to accept the default option of using all the remaining space.

Use the `'p'` command to inspect the partition table. Finish the configuration with `'w'`. Only then, the above configurations will be applied to the storage device.

In certain situations, you may need to eject the device and reconnect it, so that the system recognizes the new partition table. If the partition list obtained with `lsblk` does not match the expected, you should try that procedure.

6.5) The first partition should contain the VFAT file system contained in the image file `vfat.img`. The exact copy can be performed with the following command:

```
dd if=vfat.img of=/dev/sdb1
```

6.6) Mount `/dev/sdb1` and inspect the `cmdline.txt` file:

```
mkdir m1
mount /dev/sdb1 m1
cat m1/cmdline.txt
```

This file contains parameters that are passed to the Linux kernel at boot time. Using a text editor, change the file contents to exactly the following:

```
dwc_otg.lpm_enable=0 console=serial0,115200 root=/dev/sda2
rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait
```

The key parameters are marked in bold above. The **root** parameter indicates the partition containing the root file system. The **rootwait** parameter tells the kernel to wait until /dev/sda2 is detected. This is sometimes necessary, because some USB devices need some time to initialize and, therefore, to be detected by the kernel. If this does not work, use the **rootdelay=n** parameter (where n is the number of seconds to wait) instead.

The second partition, which will be used to store the root file system created by Buildroot, should be of type EXT4. The empty file system is created as follows:

```
mkfs.ext4 /dev/sdb2
exit
```

Creation of the root file system and final tests

7) **Back to the initial terminal**, check if the compilation has already finished. The next step will be testing the generated root file system (`output/images/rootfs.ext4`) using QEMU. QEMU is able to emulate a machine with the same processor as Raspberry Pi but with some different hardware components. Due to this, it cannot use the Raspberry Pi Linux kernel. Instead, just for testing purposes, we will use the following kernel, obtained from https://github.com/dhruvvyas90/qemu-rpi-kernel:

```
cd ~/student_number/Lab3
wget http://ave.dee.isep.ipp.pt/~jes/arcom/Lab2-QEMU/kernel-qemu-4.14.50-stretch
wget http://ave.dee.isep.ipp.pt/~jes/arcom/Lab2-QEMU/versatile-pb.dtb
```

Finally, the command line to launch QEMU is the following:

```
qemu-system-arm -cpu arm1176 -m 256 -M versatilepb \
-kernel kernel-qemu-4.14.50-stretch -dtb versatile-pb.dtb \
-append "root=/dev/sda" \
-drive file=buildroot-rpi/output/images/rootfs.ext4,format=raw
```

When asked for the login credentials, login as user `root` with password `root`. You will be presented the shell command prompt (#). Check the directory structure, the memory and storage usage and close the emulator:

```
cd /
ls
free
df
poweroff
```

8.1) **Back in the initial terminal**, upload the Buildroot config file (`.config`) to your account at ave.dee.isep.ipp.pt:

```
sftp student number@ave.dee.isep.ipp.pt
mkdir WWW/arcom
cd WWW/arcom
put buildroot-rpi/.config buildroot-config-rpi
quit
```

8.2) Run `make sdk` to generate an archive containing the toolchain files, for later use. The archive is created in the Buildroot subdirectory `ouput/images`.

9) The next step consists of creating the root file system in the USB device. Copy the contents of `output/images/rootfs.ext4` to the file system in the second partition. Assuming, as before, that your device is identified as the second "`sd`" storage device (`/dev/sdb`):

```
su
mkdir m
mkdir m2
mount buildroot-rpi/output/images/rootfs.ext4 m
mount /dev/sdb2 m2
cp -a m/* m2/
umount m m2
```

10) Test the Linux distribution stored in your USB device both in QEMU (by setting `file=/dev/sdb2` in the `-drive` parameter), and in the Raspberry Pi board.

11) Compress the `rootfs.ext2` file and upload the resulting file to your web area at ave.dee.isep.ipp.pt:

```
gzip -k buildroot-rpi/output/images/rootfs.ext2
scp buildroot-rpi/output/images/rootfs.ext2.gz \
  student_number@ave.dee.isep.ipp.pt:WWW/arcom/
```

Resize the first partition

12) In case you own a large USB storage device (greater than 2 GiB) it might be useful to make the first partition larger. This way, you can also use the device to store your files when using Microsoft Windows machines.
If that is the case, repeat the procedure above using a larger size for the first partition. Instead of making the byte-for-byte exact copy of `vfat.img` to `/dev/sdb1` (using `dd`), you should create the VFAT filesystem on the first partition (`mkfs.vfat /dev/sdb1`) and copy the contents of the file system in `vfat.img` to that partition, analogously to what was done in point 8.

Appendix

QEMU provides partial emulation specifically for the Raspberry Pi 2 (`qemu-system-arm -M raspi2`) and Raspberry Pi 3 (`qemu-system-aarch64 -M raspi3`) boards. However, the current implementation does not support USB emulation, which, among other aspects, impacts the network support.

However, it might still be useful to test basic functionality of code targeted specifically to the Cortex A-57 processor architecture.

.

In order to use this type of emulation, some firmware files are required. These files are the Linux kernel image and a dtb file (Device Tree Blob/Binary). The latter contains information regarding Raspberry Pi hardware components. There files are typically present in the first partition of the system images, and can also be obtained from the Github

Assuming the root file system is contained in `rootfs.ext4`, the command line to launch QEMU comes as follows:

```
qemu-system-arm -M raspi2 \
-kernel kernel7.img \
-dtb bcm2709-rpi-2-b.dtb \
-sd rootfs.ext4 \
-append "rootdelay=1 loglevel=1 root=B300 console=ttyAMA0" \
-serial stdio
```

The code B300 in the kernel parameter line (`-append` parameter) corresponds to the Secure Digital (SD) card reader (`/dev/mmcblk0`). The file passed to the `-sd` parameter is used to emulate a SD card.

The text console of the emulated machine will be presented directly in your terminal.

Appendix

From the buildroot online user manual (2018/09):

Mandatory packages:

Build tools:

- `which`
- `sed`
- `make` (version 3.81 or any later)
- `binutils`
- `build-essential` (only for Debian based systems)
- `gcc` (version 4.4 or any later)
- `g++` (version 4.4 or any later)
- `bash`
- `patch`
- `gzip`
- `bzip2`
- `perl` (version 5.8.7 or any later)
- `tar`
- `cpio`
- `python` (version 2.6 or any later)
- `unzip`
- `rsync`
- `file` (must be in `/usr/bin/file`)
- `bc`

Source fetching tools:

- `wget`

Optional packages

Configuration interface dependencies:

For these libraries, you need to install both runtime and development data, which in many distributions are packaged separately. The development packages typically have a -dev or -devel suffix.

- `ncurses5` to use the `menuconfig` interface
- `qt4` to use the `xconfig` interface
- `glib2`, `gtk2` and `glade2` to use the `gconfig` interface

Source fetching tools:

In the official tree, most of the package sources are retrieved using wget from ftp, http or https locations. A few packages are only available through a version control system. Moreover, Buildroot is capable of downloading sources via other tools, like rsync or scp (refer to Chapter 19, Download infrastructure for more details). If you enable packages using any of these

methods, you will need to install the corresponding tool on the host system:

- `bazaar`
- `cvs`
- `git`
- `mercurial`
- `rsync`
- `scp`
- `subversion`

Document history

- 2018-10-04 – minor updates by Jorge Estrela da Silva (jes@isep.ipp.pt)
  - Added points 5.1, 8.1, 8.2
  - Update to `kernel-qemu-4.14.50-stretch`
  - New `vfat.img`, from Raspbian-stretch (kernel 4.14.50). Fdisk configuration changed.
  - Reference to Raspberry Pi 3 in the Appendix
- 2017-10-01 – created by Jorge Estrela da Silva (jes@isep.ipp.pt)