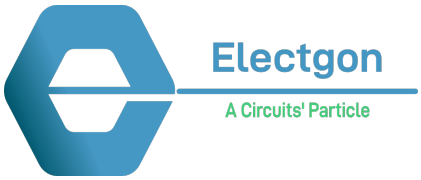

Ubuntu Snappy

Emulation Using QEMU

ELECTGON
www.electgon.com
ma_ext@gmx.net

15.04.2018

Contents

Abstract

Developing a Snappy based operating system is important for embedded Linux developments, specially for engineers who target to work on ARM processors. This tutorial is going to show how to build Snappy using QEMU emulation based on ARM processor. Some introductory is needed first to understand mechanism and structure of Snappy, followed by guide on how to work with snappy using QEMU emulation.

1 Introduction to Snappy

Many Embedded Systems depend currently on embedded Linux since any Linux system can be used for embedded applications. Snappy is a Linux distribution that arose to meet Internet of Things demands. Released in December 2014, Snappy can be considered as an Ubuntu core that runs together with framework and applications. What is new about this core is it can update the core and its applications. Updates are received from Ubuntu cloud and - according to Snappy Ubuntu website – your system has a backup in the cloud before update process so that you can rollback to previous state under any circumstances. Thus you can find your system exists in the cloud anytime, that is why Snappy is said to be matching internet of things architecture.

Snappy provides new mechanism of managing applications, any application is contained in its own folder. This includes all packages, files needed for the application to run. This means that each application is independent on the others because no shared packages are common to the applications.

Nevertheless, Snappy has its own management for updating and installing new packages. I mean although it is an Ubuntu core, however no apt-get is available in snappy. To install new package you need to use command like sudo snappy install <package name>.

Applications in that sense are considered as 'contained' in a box. Frameworks are needed to provide runtime for these application. Using these frameworks you can run your application on any operating system if the operating system supports this framework. For instance Java is a framework that is supported in almost all known desktop operating systems. Snappy supports a framework called Docker.

2 Emulation with QEMU

Snappy is announced as a devices (or Things) OS. i.e. it is not desktop OS (maybe in future it becomes so). So to experience it, emulation is a good method. In this tutorial we will see how to run this emulation based on ARM processor. Notice that ARMv7 processors are latest processors architecture that Snappy can work on i.e. ARMv6 can't run Snappy.

2.1 Kernel Settings

Since Snappy is just Ubuntu core (i.e. rootfs) it needs a kernel and bootloader. At time of writing this tutorial, linux 3.18 was the latest kernel that supports Snappy so we will use it for the emulation. We will boot this system on ARM versatile express machine as this machine is based on ARMv7.

2.2 Prepare Sources

QEMU: in previous tutorial we discussed how to install and work with QEMU, so please refer to it. Cross-compiler: Also was discussed before in previous tutorial. We are going to use codesourcery. Linux Kernel: Navigate to Linux kernel home page and download

linux 3.18 kernel
https://www.kernel.org/pub/linux/kernel/

Snappy: I have used Snappy image that works with ARM processors from Raspberry website
http://www.raspberrypi.org/downloads/
Another version of Snappy image is available in this link (download pi2.img file)
http://people.canonical.com/~lool/pi2-device-and-oem/

2.3 Start Configuration

We are going now to configure downloaded kernel to work on ARM machine. Unpack your downloaded kernel

```
$tar xvf linux-3.18.10.tar.bz2
$cd linux-3.18.1
$make ARCH=arm vexpress_defconfig
$make ARCH=arm menuconfig
```

Configuration menu shall start. You should set configuration according to table 1

Section	Feature	Sub-Feature	Subsub-feature	Action
General Setup	Cross-compiler tool prefix			set its value to Arm-none-linux-gnueabi-
Enable Block Layer	Support for large (2TB+) block device and files			Enable
	Block layer SG support v4			Enable
	Block layer SG support v4 helper lib			Enable
System Type	ARM system type	Allow multiple platforms to	be selected	Enable
	Support ARM V6 processor			Enable
	Support ARM V7 processor			Enable
	Enable ThumbEE CPU extension			Enable
	ARM errata: Invalidation of the Instruction Cache operation can fail			Enable
	ARM errata: Possible cache data corruption with hit-under-miss enabled			Enable
Bus Support	PCI Support			Enable
Kernel Features	Use ARM EABI to compile the kernel			Enable
	Allow old ABI binaries to run with this kernel			Enable
Floating point emulation	VFP-format floating point maths			Enable
Device Drivers	SCSI Device Support	SCSI Device Support		Enable
		SCSI Disk Support		Enable
		SCSI CDROM support		Enable
		SCSI low-lever drivers	SYM53C8XX Version 2 SCSI support	Enable
	Generic Driver Options	Maintain a devtmpfs filesystem to mount at /dev		Enable
		Automount devtmpfs at /dev, after the kernel mounted the root		Enable
	Input device support	Event interface		Enable
File systems	Ext3 journalling file system support			Enable
	The Extended 4 (ext4) filesystem			Enable
	Pseudo filesystems	Virtual memory file system support (former shm fs)		Enable

Table 1: Kernel Configurations

After this configuration step, your kernel is ready for compilation.

```
$make ARCH=arm
```

Up to here you have compiled successfully a kernel for emulation usage. You can find an image created for this kernel in the directory arch/arm/boot, you can find there zImage of the kernel. We will use this image to run the emulation, so take a copy and place it in directory which contains the pi-snappy.img file

Now navigate to this directory and run the following command

```
$qemu-system-arm -kernel vexpress_04_zImage -append "root=/dev/mmcblk0p2 rw" -m
     1G -cpu cortex-a9 -M vexpress-a9 -sd pi-snappy.img
```

In this command “vexpress_04_zImage” is my kernel image name. “/dev/mmcblk0p2” is the partition on which booting process will start. “cortex-a9” is the processor that is running on the machine “vexpress-a9” which is assumed to act as Raspberry Pi2.

3 Running the Emulation

After running previous command, Snappy shall boot. When it comes ready, you can login with the following credentials

username: ubuntu

password: ubuntu

You can start working with Snappy. If you faced problems in internet connection. Try to debug this problem by first recognize if your emulation can detect ethernet card or not by running this command dmesg | grep eth you should found this card is available.

Next check if you got IP address or not by running

```
ifconfig eth0
```

If no ip address was assigned to your emulation machine, you fix that by adding the following lines to interface file

```
$cd /etc/network
$vi interfaces
```

```
auto eth0
iface eth0 inet dhcp
```

To edit using vi, go to append mode by typing ’a’ after last part then press enter then add the previous lines. To save in vi go back to command mode by pressing esc then type :wq then enter.

It should work now, check by ping on any website ping www.google.com

If everything is working fine, you can start building your work in Snappy. To learn working with Snappy you can follow guide in Snappy website

https://developer.ubuntu.com/en/snappy/tutorials/using-snappy/

Bibliography

[1] http://www.nico-maas.de/wordpress/?p=1061

[2] http://thenewstack.io/snappy-ubuntu-core-powering-microcontrollers-to-microservices/

[3] https://developer.ubuntu.com/en/snappy/start/

[4] http://people.canonical.com/~lool/pi2-device-and-oem/

[5] http://www.tuxradar.com/content/diagnose-and-fix-network-problems-yourself

[6] http://learn.linksprite.com/pcduino/first-view-snappy-ubuntu-core-for-pcduino3