Qemu Support in Slackware ARM

This document describes the process of installing **Slackware ARM** inside of QEMU.

- QEMU is a full system emulator that can emulate a range of real hardware architectures. In this case we will use QEMU to emulate the ARM Ltd. “Versatile Express” development board.
- QEMU provides a platform to allow an operating system to believe that it is running on real hardware.
- QEMU runs on your Slackware PC or server. In most cases this will be a desktop machine. However, it is possible to run QEMU on a headless server and use VNC for graphics.

Who is Slackware ARM in QEMU aimed at?

The aim of installing Slackware ARM inside QEMU is to allow people who do not have ARM hardware to try out Slackware ARM.

While QEMU is an excellent emulator, it is very slow compared to real ARM hardware. QEMU was used for approximately a year to develop most of Slackware ARM version 12.2. Due to the slow speed of QEMU, distcc can be used on several x86 machines to speedup compilations. It's possible to effectively use Slackware ARM in QEMU, however be aware that it will not present the best user experience due to its lack of speed. Ideally Slackware ARM should be run on real ARM hardware.

This document is a work in progress and it targets the development release of Slackware ARM, Slackwarearm-current.

Documentation for Slackware ARM 14.0, 14.1, and 14.2, can be found here:

- Slackwarearm-14.0
- Slackwarearm-14.1
- Slackwarearm-14.2

Installation Environment Assumptions

Several assumptions are made to aid in writing this document.

- The host machine is running a full installation of Slackware x86 or x86_64
- You have root access on the host machine and can compile software
- The Slackware host and QEMU emulator are on a secure Local Area Network
- You can export a Network File System file share on the host machine that shares the Slackware ARM tree with the QEMU guest
- The root directory of the exported NFS share on the host is located at /export

These guidelines can be met with one Slackware x86/x86_64 host machine. This single host machine can serve the NFS content, as well as host the Slackware ARM installation inside QEMU.

Slackware x86 Host Prerequisites

1. Download the **Slackwarearm** distribution package tree with rsync
2. Populate a location with the Slackware ARM QEMU files
3. Configure the Network File System (NFS) on the Slackware x86 Host
4. Install QEMU and the device-tree-compiler software on the Slackware x86 host
5. Configure QEMU Permissions on the Slackware x86 Host
6. Create the QEMU disk image using the helper scripts

Download Slackware ARM

Download Slackware ARM to your Slackware x86 Host. In this tutorial rsync is used to mirror Slackware ARM. Before running rsync, make your user has read, write, and execute permissions within the /export directory.

```
mkdir -p /export/slackwarearm
cd /export/slackwarearm
rsync --exclude '*/source/*' --delete -Pavv
ftp.arm.slackware.com::slackwarearm/$SLACKREL .
```

If you wish to use a web browser, wget, or lftp to download Slackware ARM, the mirror is here.

The $SLACKREL variable refers to the Slackware ARM tree you downloaded. If you chose Slackwarearm-14.2, the rsync URL will be ftp.arm.slackware.com::slackwarearm/slackwarearm-14.2

Populate Slackware ARM Files and Directories

In order to boot the Slackware ARM installer you need to create a directory that stores the Kernel and Initial RAM disk. You also need a few helper scripts to run QEMU. In this totorial, all of these files will be stored in /export/armhost.

Copy the kernel and RAM disk:

```
mkdir -p /export/armhost
cp -fa /export/slackwarearm/$SLACKREL/kernels/armv7/{zImage*,initrd*,dtb}
/export/armhost/
cp -fa /export/slackwarearm/$SLACKREL/isolinux/initrd-armv7.img
/export/armhost/
```

Download and copy the QEMU helper scripts:

```
# cd /tmp ; mkdir qemu ; cd qemu
# rsync -Prlvv --delete
ftp.arm.slackware.com::slackwarearm/boardsupport/qemu/$SLACKREL .
# cd $SLACKREL
# cp -fav helper-scripts/* /export/armhost/
```

You can find the helper scripts here if you do not wish to use rsync to download them.

The $SLACKREL variable refers to the Slackware ARM tree you downloaded. If you chose Slackwarearm-14.2, the path will be /export/slackwarearm/slackwarearm-14.2. The rsync URL for the QEMU helper scripts will be:

```
ftp.arm.slackware.com::slackwarearm/boardsupport/qemu/slackwarearm-14.2
```

Configure Network File System services

The Slackware x86 host needs to be configured appropriately in order to run the Slackwarearm installation process within QEMU. The easiest and most efficient way to do so is by setting up a Network File System (NFS) share on the host machine. This NFS share will be used by the Slackwarearm client (running inside QEMU) to access the Slackware distribution package tree on the host. NFS services are not the only way to serve the Slackware distribution package tree to the QEMU client. Other methods are quite a bit slower when used with QEMU emulated networking. As a result, the NFS service is used for this tutorial.

You need to know the QEMU network settings in order to access the NFS share. It is recommended that you use NAT mode networking with QEMU. NAT mode allows direct access to the Slackware x86 host NFS share through a QEMU hosted virtual network. More information about the virtual network and assigned addressing is discussed later on in this guide.

With a text editor, as root, add the following to the Slackware host's /etc/exports:

```
# QEMU guest virtual IP address
/export/slackwarearm
10.0.2.15(insecure,ro,nohide,root_squash,sync,no_subtree_check)
```

NAT mode does not allow direct access to the physical Local Area Network. You will need to set up a bridged network interface if you decide you need direct network access. NAT mode will be sufficient for most users.

Use something similar to the following if you plan to use a bridged network interface:

```
/export/slackwarearm
xxx.xxx.xxx.x/255.255.255.0(insecure,ro,nohide,root_squash,sync,no_subtree_c
heck)
```

Replace xxx.xxx.x.x/255.255.255.0 and match it with your network configuration. (Example: 192.168.1.0/255.255.255.0)

Activate the NFS share by executing the following as root:

```
# chmod +x /etc/rc.d/{rc.rpc,rc.nfsd}
# /etc/rc.d/rc.nfsd start
# exportfs -va
```

Install QEMU and device-tree-compiler

Installing SlackBuilds is not apart of the scope of this article. If you need assistance with installing QEMU or device-tree-compiler, please refer to the SlackBuilds.org HOWTO page. With that said, there are a few recommendations:

- If you are running Slackware-current you can install QEMU and skip installing the device-tree-compiler package. Slackware-current already includes the device-tree-compiler package in a full installation.
- Please be certain that you did a **full** Slackware installation on your x86 host prior to installing these SlackBuilds.
- Slackware 14.0, 14.1, and 14.2 users need to install the device-tree-compiler package from SlackBuilds.org prior to installing QEMU.
- You can download and install QEMU from SlackBuilds.org.
- You will not have the ability on 14.0, 14.1, and 14.2 systems to emulate the ARM architecture in QEMU if you do not first install the device-tree-compiler package.

QEMU Permissions

There are a few permissions that need to be set once you have successfully installed QEMU on your system. The QEMU client will be launched by running the /usr/bin/qemu-system-arm binary. This binary needs root permissions in order to be executed. You can run this binary with sudo by editing /etc/sudoers. Using sudo is the most secure option if you have multiple users on your system. If you are the only user on your system then setting the setuid permission as root is sufficient. Adjusting these permissions will allow a normal user to configure and boot the QEMU guest without logging in as root.

Setting the setuid root permission requires that you log in as root. As root execute the following commands:

```
# chmod +s /usr/bin/qemu-system-arm
```

If you plan to configure QEMU to use a network bridge, you also need to set the setuid root permission for /sbin/ifconfig and /sbin/brctl.

```
# chmod +s /sbin/{ifconfig,brctl}
```

Create QEMU Disk Image

Prior to booting the Slackware ARM installer in QEMU, you must create a disk image that acts as an emulated SD Card. This disk image is used to emulate the MMC controller in Slackware ARM. Earlier you copied the Slackware ARM helper scripts to /export/armhost/. Within this directory there is a script, **makeimg**. This script creates a 15GB disk image automatically in /export/armhost/ when it is executed. Initially all you need to do is run this script.

Switch to the directory where you copied the helper scripts and execute **makeimg**:

```
cd /export/armhost
./makeimg
```

For reference, this is the **makeimg** script:

```
# Create the QEMU disk image - the emulated SD card.

IMG=sdcard.img
SIZE=15G

rm -f $IMG
qemu-img \
   create \
   $IMG $SIZE
```

Be aware that once you have installed Slackware ARM onto this disk image you must move it to a different directory for storage, or you risk it being destroyed when **makeimg** is executed at a later time.

QEMU Network Settings

This section covers the process of setting up QEMU guest networking. Two different processes will be described. *Network Address Translation mode (NAT)* is the first and recommended way to get a functional network in QEMU guests. The second is *bridged mode*. NAT mode does not allow direct access to the Slackware x86 host's physical network and bridged mode does. It is best to use bridged mode if you plan on doing any more advanced network operations that require full access to the host and the host's physical network. **Most users will want to use NAT mode.**

There are many different ways to configure QEMU guest networking. This document will only cover the QEMU functionality required to boot Slackware ARM. Refer to the QEMU man pages or the QEMU documentation if you need further explanation.

Later on when you boot the Slackware ARM installer in QEMU you may need to modify the **txqueuelen** for your network interfaces. This is because large Slackware packages time out while being downloaded from the NFS share on the host. This happens because QEMU emulation is very slow. The NFS daemon on your host machine occasionally shuts down the network socket before large packages (rust, kernel-firmware, etc) finish being copied to the SD Card. Setting the txqueuelen to **10000** for all network interfaces should be sufficient to prevent this anomaly. The following command seems to resolve this issue:

```
ip link set eth0 txqueuelen 10000
```

Run this command for each network interface actively used by QEMU.

QEMU NAT Mode Networking

NAT mode does not require any additional configuration on the Slackware x86 host machine or in the QEMU guest machine. Here is a shortened example of a QEMU guest being launched with NAT mode networking:

```
# cd /export/armhost
# qemu-system-arm -nographic \
   -m 1024 \
   -M vexpress-a9 \
   -k en-us \
   -net nic \
   -net user
   ..snip..
```

The **-net nic** and **-net user** options enable QEMU to start the Slackware ARM guest with NAT mode networking enabled. These settings are documented further in both the **installer_launch** and **disk_launch** helper scripts.

With NAT mode enabled, QEMU launches a virtual network of 10.0.2.0/24. The QEMU guest will be assigned the IP address 10.0.2.15. The guest can access the Slackware x86 host at 10.0.2.2 and the QEMU DNS server runs at 10.0.2.3. QEMU does not have direct access to the host's Local Area Network. This means that the QEMU guest isn't assigned a physical IP address by your router DHCP service. You will not be able to ping the QEMU guest from the host machine but the guest should be able to ping the host machine at 10.0.2.2. The QEMU guest should be able to access the internet and communicate with the host machine.

QEMU Bridged Mode Networking

The best way to set up a bridged network interface for QEMU is with the provided helper script, rc.local-additions. If you choose to take this route, you need to disable the NetworkManager service. NetworkManager comes with Slackware, but it is not developed by Slackware. As a result, the process of configuring a bridge with NetworkManager is not supported in this guide.

You can find the helper script with comments online here: rc.local-additions

The following commands must be executed as root to disable NetworkManager:

```
# /etc/rc.d/rc.networkmanager stop
# chmod -x /etc/rc.d/rc.networkmanager
```

The following network settings are assumed for the **Slackware x86 host machine**. Adjust these values throughout the remainder of this guide if you use different network settings.

```
Default Gateway: 192.168.1.1
Static IP address: 192.168.1.2
Network Mask: 255.255.255.0 / 192.168.1.0/24
Name server: 192.168.1.1
```

You need to edit the *qemu-network-tun.sh* helper script for QEMU. It is required to bring up the QEMU guest network interface. It should exist in /export/armhost. Change the IP address listed in $BRIDGEIP to match your network settings.

File: /export/armhost/qemu-network-tun.sh

```
#!/bin/sh

# This is the IP of 'tap0' on the Slackware/x86 host:
BRIDGEIP=192.168.1.4

modprobe tun
/sbin/ifconfig $1 $BRIDGEIP netmask 255.255.255.0
/sbin/brctl addif br0 $1
```

Here is the relevant portion of the rc.local-addtions script that requires modification. This script is meant to replace on the host machine /etc/rc.d/rc.local. Changes depend on your network settings:

```
.. snip ..

# Turn on the bridge.  Note that this is a different IP from
# the one specified in your qemu-network-bridge.sh script
# in your 'armhost' directory on your Slackware x86 box.
# You need a bridge IP, a tunnel (tap0) IP, and then another
# IP which is assigned to the Slackware ARM host (by Linux inside QEMU)
# to its own eth0.
# I tried bringing this up after eth0 but the bridge didn't work.
# I don't know why that is!
ifconfig br0 192.168.1.3 up

# Put back the original IP for eth0:
ifconfig eth0 192.168.1.2 up

#
route del default
route add default gw 192.168.1.1
```

These network settings assume static IP addressing. The br0 interface is the bridged network interface. The eth0 interface is the host machine network interface that allows the host machine to retain network connectivity. The default gateway, 192.168.1.1, typically points to the network gateway on the physical network.

Copy rc.local-addtions to /etc/rc.d/rc.local once you finish editing it. Then mark it executable.

```
# cp /path/to/rc.local-additions /etc/rc.d/rc.local
# chmod +x /etc/rc.d/rc.local
```

At this point it is recommended reboot your Slackware x86 host to assure the settings in rc.local are in use and that NetworkManager is completely disabled.

You may need to edit the /etc/resolv.conf file on the host. Add in the IP addresses of your preferred primary and secondary name server(s) since you are not receiving these IP addresses by other means. The name server is 192.168.1.1 (or the default gateway) in this tutorial

Modify the launcher helper scripts once you are certain your host has the appropriate network

settings. Pay close attention to the **$NETTYPE** variable. Details about how to use this variable are documented in both the **installer_launch** and **disk_launch** helper scripts. Edit the **$MACADDR** variable for each QEMU instance if you are running more than one Slackware ARM guest at once.

Install Slackware ARM

I will assume that you are now in X Windows, running as your normal user account, and that you followed the steps outlined earlier in this document. As stated earlier, QEMU runs extremely slow when emulating the ARM architecture. Depending on your hardware set up it may take several hours or more for the Slackware installer to copy all packages to the emulated disk. If the Slackware installer appears to be unresponsive, check your system process monitor (top or htop) to see if the QEMU process is still active. A good sign that QEMU is still active is that a single CPU core is operating at 100 percent.

The Slackware ARM installer is mostly identical to the Slackware x86 installer. There is no learning curve to install Slackware ARM if you have installed Slackware before.

Booting the Installer

In order to boot the installer you will need to configure and execute the **installer_launch** script within a terminal window.

```
cd /export/armhost
./installer_launch
```

The **installer_launch** script can be found here.

You will see some warnings from QEMU about being unable to open audio and video devices. Those warnings can safely be ignored. Next you will see the Linux kernel boot messages and eventually the installer asking about what key map you want to use. Once you select your key map and log in to the system you will notice that the installer obtains an IP address via DHCP. The DHCP assigned IP address and resulting network configuration relies heavily on how you set up your networking on the host machine. If QEMU does not assign an IP address to the guest, then you need to go back and verify your network settings are configured appropriately.

Partitioning

The emulated SD Card created with the **makeimg** command is a blank image. You will have to partition this SD Card with the installer. It is best to keep the partition scheme simple in our case. It is recommended that you create a 200MB swap partition and to allocate the rest of the disk to the root partition. You can use the **fdisk** or **cfdisk** tools to create the partitions.

Example partitioning scheme:

```
/dev/mmcblk0p1 - 200MB swap
/dev/mmcblk0p2 - the rest of the disc, "Linux" - type 83.
```

Setup and Configuration

Run the **setup** command at the shell prompt after you exit the partitioning tool. Make the installer aware of your swap parition and root partition. It is recommended that you select the ext4 file system when you format the root partition. Next you will be prompted to select the source media. Choose option 4, **Install from NFS (Network Filesystem)**. Enter the IP address of your Slackware x86 host. Enter the path to the NFS mounted share.

The full path of the NFS share is required:

```
Enter the IP address: 192.168.1.2 # Host machine IP address
Enter the directory: /export/slackwarearm/$SLACKREL/slackware
```

Following that, you will be prompted for package selection. Slackware ARM has all of the standard Slackware packages apart from those which are x86 only. It is highly recommended that you do a **full** installation to satisfy all system dependencies. Please be patient, this is the most time consuming part of the installation process.

After installation has finished, running 'MKFONTDIR AND MKFONTSCALE UPDATE' takes a long time.

At the Network Setup screen it is best to select the DHCP option. The DHCP option best compliments QEMU's NAT mode and both bridged mode networking options. The only reason not to select DHCP is if your physical network uses static IP addressing.

Next you will reach the Window Manager selection for the X Windows server. It is recommended that you select a light weight window manager, such as Fluxbox or WindowMaker. KDE and Xfce are not very useful within the QEMU guest due to speed constraints.

Post-Installation

Once you complete the installation process you should drop to a shell prompt to configure the SSH Daemon. By default, OpenSSH does not allow root to log in with a password. This is a security concern. You may want to think about this carefully if your device is connected directly to an untrusted network. It is best to make a user account for remote connections to the SSH service and escalate privileges locally with “su” or “sudo”. If you wish to use root to log in remotely, follow these steps:

1. Opt to drop in to a 'shell' when you exit from the installer
2. At the shell, enter:

```
# sed -i 's?^#PermitRootLogin.*?PermitRootLogin yes?g'
/mnt/etc/ssh/sshd_config
# poweroff
```

This completes the installation process of Slackware ARM within QEMU.

Boot Slackware ARM with QEMU

Congratulations for making it this far! The next step is booting into your fresh installation of Slackware ARM. Locate the **disk_launch** helper script in /export/armhost and modify it to fit your needs.

```
# cd /export/armhost
# vi disk_launch
```

This script has a few variables you may want to change.

- **ROOTFSTYPE** - root file system type, ext4 is the default
- **ROOTFSDEV** - location of the root partition within the SD Card image
- **KEYBOARD** - keyboard locale you wish to use, typically the same as what you chose during installation
- **NETTYPE** - network configuration, NAT mode or bridged mode

The **disk_launch** script for Slackwarearm-current can be found online, here.

The first boot will take quite a while. This is due to the fact that Slackware will generate the font cache for the first time. Start up QEMU by executing the disk_launch script.

```
# ./disk_launch
```

Assuming all is well, you can begin using Slackware ARM just as you would any other Slackware installation.

Slackware ARM Graphical User Interface

Work in Progress

This section will discuss the positives and negatives around running X Windows, which window manager or desktop environment to use, and the ways you can start it up.

Sources

- Originally written by Stuart Winter
- Original source: http://ftp.arm.slackware.com/slackwarearm/boardsupport/qemu/

- Modified and Maintained by Brenton Earl (mralk3)

howtos, hardware, arm, user mralk3