



# Course Title: Working with Qemu

Faculty: Rajesh Sola, Dr. Vivek Kaundal, Bharath G & Bhargav N



L&T Technology Services



LTTS

**GLOBAL  
ENGINEERING  
ACADEMY**





# Mastering Embedded Linux with Qemu

Faculty: Rajesh Sola, Dr. Vivek Kaundal



L&T Technology Services



LTTS

GLOBAL  
ENGINEERING  
ACADEMY





# Mastering Embedded Linux with Qemu

Topic : Working with Qemu



L&T Technology Services




LTTS

GLOBAL  
ENGINEERING  
ACADEMY



# Agenda

---



Setting up Qemu

Building Custom Kernel

Cross Compilation Techniques

Bootimg Techniques

# Version

| Version | Reviewed by | Approved by | Remarks |
|---------|-------------|-------------|---------|
| 1.0     |             |             |         |





# Qemu Based Emulation



# Clean Workspace

---

- “ Choose a clean directory for all the work,
- “ You may choose directory like workspace/eworkspace/kworkspace/ebuildws/ews or with any sensible name under home directory
- “ Don't use Desktop, Downloads, Documents, Music, Videos, Pictures etc, which are meant for other purpose.
- “ Avoid spaces or special symbols in path names
- “ Under this workspace keep different sub directories for downloaded packages, extracted source to build, configuration files, examples etc.

# Setup Qemu

// Install Qemu, a full system emulator for ARM target architecture

```
sudo apt install qemu-system-arm
qemu-system-arm -v
qemu-system-arm -M ?
qemu-system-aarch64 -v
```

// Alternative – Build from sources

```
Download latest stable version of Qemu source from
https://www.qemu.org/download/#source
tar -xvf qemu-5.x.y.tar.xz # replace 5.x.y by suitable version
cd qemu-5.x.y
./configure --targetlist=armsoftmmu, armlinuxuser
                                     --enable-sdl  --prefix=/opt/qemu-5.x.y
make && make install
export PATH=/opt/qemu-4.x.y/bin:$PATH # you may add this to ~/.bashrc
```



# Rootfs

// Download core-image-minimal-qemuarm.ext4 from

<http://downloads.yoctoproject.org/releases/yocto/yocto-2.5/machines/qemu/qemuarm/>

// Rename core-image-minimal-qemuarm.ext4 as rootfs.img

// Align the size of rootfs

```
e2fsck -f rootfs.img  
resize2fs rootfs.img 16M
```

// Alternative

```
# Download core-image-minimal-qemuarm.tar.bz2 from same link  
qemu-img create -f raw rootfs.img 64M  
mkfs.ext4 rootfs.img  
mount -o loop,rw,sync rootfs.img /mnt/image # mkdir for first time  
tar -jxvf core-image-minimal-qemuarm.tar.bz2 -C /mnt/image  
umount /mnt/image
```

# Toolchain

## // Install linaro toolchain from ubuntu package manager

```
sudo apt install gcc-arm-linux-gnueabi          # soft float
sudo apt install gcc-arm-linux-gnueabihf        # hard float
```

// We'll go for soft float for now, due rootfs compatability

## // Alternatively, download latest pre-built linaro toolchain from as per host architecture

// From <https://releases.linaro.org/components/toolchain/binaries/latest-7/arm-linux-gnueabi/>, say v7.5.0

```
tar -xvf gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabi.tar.xz -C /opt
export PATH=/opt/gcc-linaro-linux-gnueabi-7.5.0-2019.12_linux/bin:$PATH
```

// Similarly gcc-linaro-7.5.0-2019.12-x86\_64\_arm-linux-gnueabihf.tar.xz for hard float

# Your First Boot (Emulation)

- // Collect prebuilt zImage, vexpress-v2p-ca9.dtb from faculty
- // Ensure rootfs.img is also in same location
- // Emulate using Qemu – sdcard approach

```
qemu-system-arm -M vexpress-a9 -m 1024 -serial stdio \  
-kernel zImage -dtb vexpress-v2p-ca9.dtb \  
-sd rootfs.img -append "console=ttyAMA0 root=/dev/mmcblk0 rw"
```

- // Emulate using Qemu – initrd approach

```
qemu-system-arm -M vexpress-a9 -m 1024 -serial stdio \  
-kernel zImage -dtb vexpress-v2p-ca9.dtb \  
-initrd rootfs.img -append "console=ttyAMA0 root=/dev/ram0 rw"
```

# First Steps on Target

---

```
uname -r  
uname -v  
uname -a  
cat /proc/cpuinfo  
free -m  
df -kh  
mount  
dmesg
```

# Building Custom Kernel (Qemu)



# Download Kernel Source

- // Download any recent LTS version of kernel source
- // Let's go with 4.14.x for now, for better compatibility with Qemu

```
wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.14.202.tar.xz  
tar -xvf linux-4.14.202.tar.xz
```

- // Or you can checkout kernel source from git.kernel.org, and switch to desired branch

```
git clone https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git  
cd linux  
git checkout tags/v4.14 -b v4.14
```

- // Let's call extracted content (or) checked out content as KSRC

# Obtain Configuration File

- “ Locate default config available in KSRC/arch/arm/configs, we'll refer vexpress\_defconfig for Versatil Express target being used for Qemu emulation
- “ Or collect any well tested configuration file as base config

```
make ARCH=arm mrproper  
make ARCH=arm vexpress_defconfig  
(or)  
# copy custom config file as .config under KSRC
```

- “ Please note that mrproper will remove built files, including the configuration. So run this only for any new build.

# Further Customization

- // Run menuconfig for further customization
- // Resolve any host dependencies at this stage, e.g. libncurses5-dev, flex, bison etc.

```
make ARCH=arm menuconfig
```

- // Let's do these minimal changes for now
  - // General Setup -> Local Version -> "-custom"
  - // Device Drivers -> Block Devices ->
    - // Enable RAM Block device support
    - // Increase default RAM disk size to suitable limit, say 65536
  - // Enable the block layer
    - // Support for large (2TB+)



# Build the kernel

// Run menuconfig for further customization

// Build kernel image

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- zImage -j <n>
```

// Build Device Tree Binaries

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- dtbs firmware
```

// Build dynamic modules (can skip for now)

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- modules
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- modules_install \
    INSTALL_MOD_PATH=<tempdir> # or mount point of target rootfs
```

# Booting with new kernel

// Collect built outcome to a temporary location

```
# switch to KSRC
cp $KSRC/arch/arm/boot/zImage      <path-of-temp-boot-dir>
cp $KSRC/arch/arm/boot/dts/*.dtb   <path-of-temp-boot-dir>
```

// Ensure rootfs.img is also in same location

// Emulate using Qemu

```
qemu-system-arm -M vexpress-a9 -m 1024 -serial stdio \
    -kernel zImage -dtb vexpress-v2p-ca9.dtb \
    -sd rootfs.img -append "console=ttyAMA0 root=/dev/mmcblk0 rw"
```

# Test the Built outcome

# In Target

```
uname -r
```

```
uname -v
```

```
ls /boot                # observe new entry
```

```
ls /lib/modules          # observe new entry
```

# In Host

```
ls -lh $KSRC/arch/arm/boot/zImage
```

```
ls -lh $KSRC/vmlinux
```



# Cross Compiling Code



# Simple Hello Module

```
#include<stdio.h>

int main() {
    printf("Hello World\n");
    return 0;
}
```

```
arm-linux-gnueabi-gcc hello.c -o h1.out
arm-linux-gnueabi-gcc hello.c -o h2.out -o static

file h1.out h2.out
ls -lh h1.out h2.out
ldd h1.out
ldd h2.out
```

```
# copy h1.out, h2.out to target rootfs
sudo mount -o loop,rw,sync rootfs.img /mnt/rootfs
sudo cp h1.out h2.out /mnt/rootfs/home/root
sudo umount /mnt/rootfs
```

# Multifile Programming

```
//test.c
int main() {
    int a,b,c,d;
    a=10,b=20;
    c=sum(a,b);
    d=square(a);
    //print c,d
}
```

```
//sum.c
int sum(int x,int y)
{
    return x + y;
}
```

```
//sqr.c
int square(int x)
{
    return x * x;
}
```

```
arm-linux-gnueabi-gcc test.c -c
arm-linux-gnueabi-gcc sum.c -c
arm-linux-gnueabi-gcc sqr.c -c
arm-linux-gnueabi-gcc test.o sum.o sqr.o \
                        -o all.out
# copy all.out to target rootfs and test
```

**TODO:-** Write a Makefile for above steps

# Static Library

```
# prepare the source code and generate .o files as earlier
arm-linux-gnueabi-ar rc sum.o sqr.o libsample.a
```

```
arm-linux-gnueabi-gcc -L. test.o -lsample -o s1.out
arm-linux-gnueabi-gcc -L. test.o -lsample -o s2.out -static
# copy s1.out, s2.out to target rootfs and test
```

```
file s1.out s2.out
# compare size of s1.out, s2.out
ldd s1.out s2.out
```

```
# check size of s2.out before strip
arm-linux-gnueabi-strip s2.out
# check size of s2.out after strip
```

- ☐ Do .a file to be shipped along with executable.
- ☐ Can we use `nm`, `objdump` on stripped executable?
- ☐ What if .a file is in one directory and test.o is in other directory?
- ☐ Necessary support from toolchain for static linking of std libraries
- ☐ Write a Makefile for above steps

# Dynamic Linking

# On Host

```
arm-linux-gnueabi-gcc -shared -o libsample.so sum.o sqr.o  
arm-linux-gnueabi-gcc -L. test.o -lsample -o d1.out  
# copy libsample.so, d1.out to target rootfs and execute
```

# On Target

```
LD_LIBRARY_PATH=. ./d1.out
```

file d1.out

```
# check size of d1.out  
ldd d1.out
```

- ☐ Do .so file to be shipped along with executable.
- ☐ What if .so file is in one directory and test.o is in other directory on host?
- ☐ What if .so file is one directory and d1.out is in other directory on target.
- ☐ Usage of LD\_LIBRARY\_PATH environment variable.
- ☐ Significance of /etc/ld.so.conf entries and ldconfig command
- ☐ Write a Makefile for all the steps



# Versioned so files

# On Host

```
arm-linux-gnueabi-gcc -shared -Wl,-soname,libsample.so \
    -o libsample.so.1.0.1 sum.o sqr.o
ln -s libsample.so.1.0.1 libsample.so
arm-linux-gnueabi-gcc -L. test.o -lsample -o d1.out
# copy d1.out to target rootfs
# copy libsample.so.1.0.1 to /mnt/rootfs/opt/mylibs
# add an entry to /opt/mylibs in /mnt/rootfs/etc/ld.so.conf
```

# On Target

```
ls /opt/mylibs
./d1.out # will give error initially
ldconfig
./d1.out
ls /opt/mylibs          # observer generated symlink
```

# Working with U-Boot



# U-Boot Sources

---

- // Download tarball from <ftp://ftp.denx.de/pub/u-boot/> and extract, choose any stable version like [u-boot-2020.10.tar.bz2](#)
- // Let's call the cloned/extracted source as USRC
- // Alternative:-
  - // Checkout U-Boot source from <https://gitlab.denx.de/u-boot/u-boot>
  - // Switch to any stable branch like v2020.10
- // You may also download offline tarball from [gitlab.denx.de](https://gitlab.denx.de/u-boot/u-boot)

# Cross Building

---

```
make ARCH=arm vexpress_ca9x4_defconfig  
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-  
# Locate generated u-boot and copy to a tempdir
```

# Simple Boot – Rootfs in SD Card

```
qemu-img create simplesd.img 64M
sudo mkfs.vfat simplesd.img
sudo mount -o loop,rw,sync simplesd.img /mnt/sdcard
# copy zImage, vexpress-v2p-ca9.dtb, rootfs.img to /mnt/sdcard
umount /mnt/sdcard
# copy simplesd.img to tempdir, where generated u-boot is copied
```

```
qemu-system-arm -M vexpress-a9 -m 1024 -serial stdio -kernel u-boot -sd simplesd.img
# Stop autoboot by hitting any key, Run the following commands in U-Boot shell
mmcinfo fatls mmc 0:0
fatload mmc 0:0 0x60200000 zImage
fatload mmc 0:0 0x60100000 vexpress-v2p-ca9.dtb
fatload mmc 0:0 0x62000000 rootfs.img
setenv bootargs 'console=ttyAMA0 root=/dev/ram0 rw rootfstype=ext4
                 initrd=0x62000000, 16777216'
bootz 0x60200000 - 0x60100000
# 16777216 is size of loaded rootfs image
# for this method ramdisk support should be enabled at kernel level (menuconfig --> Device Drivers
--> Block Devices --> RAM Block Device Support)
```

# Prepare Partitioned SD Card

```
dd if=/dev/zero of=sdcard.img bs=1M count=128
# create two primary partitions in sdcard.img using cfdisk
# Keep first partition size as small as possible, say 16M
sudo fdisk -l sdcard.img # 1048576 is 2048x512, 2048 is start of first
partition # 17825792 is 34816x512, 34816 is start of second partition
sudo losetup -o 1048576 /dev/loop20 sdcard.img
sudo losetup -o 17825792 /dev/loop21 sdcard.img
sudo mkfs.vfat /dev/loop20 sudo mkfs.ext4 /dev/loop21
sudo mount -o loop,rw,sync /dev/loop20 /mnt/boot
sudo mount -o loop,rw,sync /dev/loop21 /mnt/rootfs
#copy zImage, vexpress-v2p-ca9.dtb to /mnt/boot
# extract core-image-minimal-qemuarm.tar.bz2 to /mnt/rootfs
tar -jxvf core-image-minimal-qemuarm.tar.bz2 -C /mnt/rootfs
sudo umount /mnt/boot
sudo umount /mnt/rootfs
sudo losetup -d /dev/loop20
sudo losetup -d /dev/loop21
```

# Rootfs in partitioned SD Card

```
qemu-system-arm -M vexpress-a9 -m 1024 -serial stdio -kernel u-boot -sd sdcard.img
#Stop autoboot by hitting any key, Run the following commands in U-Boot shell
mmcinfo
fatls mmc 0:1
fatload mmc 0:1 0x60200000 zImage
fatload mmc 0:1 0x60100000 vexpress-v2p-ca9.dtb
setenv bootargs 'console=ttyAMA0 root=/dev/mmcblk0p2 rw rootfstype=ext4'
bootz 0x60200000 - 0x60100000
```

```
qemu-system-arm -M vexpress-a9 -m 1024 -serial stdio \
    -kernel zImage -dtb vexpress-v2p-ca9.dtb \
    -sd sdcard.img -append "console=ttyAMA0 root=/dev/mmcblk0p2 rw"
```

# Setup TFTP on Host

```
sudo apt install tftpd
# create /etc/xinetd.d/tftp
# with specified content
# replace server_args as per your machine
/etc/init.d/xinetd restart
```

```
service tftp
{
    protocol          = udp
    port              = 69
    socket_type       = dgram
    wait              = yes
    user              = nobody
    server             = /usr/sbin/in.tftpd
    server_args        = /* */
    disable            = no
}
```

```
sudo modprobe tun
sudo ifconfig tap0 192.168.7.1
```



# TFTP Boot on Target

```
sudo qemu-system-arm -M vexpress-a9 -m 256 -kernel u-boot -serial stdio \  
-sd sdcard.img -net nic -net tap,ifname=tap0
```

```
setenv ipaddr 192.168.7.2  
setenv serverip 192.168.7.1  
ping 192.168.0.1  
tftp 0x60200000 zImage  
tftp 0x60100000 vexpress-v2p-ca9.dtb  
setenv bootargs 'console=ttyAMA0 root=/dev/mmcblk0p2 rootfstype=ext4'  
bootz 0x60200000 - 0x60100000
```



Thank You !



L&T Technology Services

