

## 虚拟机就是开发板

笔记本： 电子与嵌入式

创建时间： 2018/7/28 12:43

更新时间： 2018/7/28 12:53

URL: <https://blog.csdn.net/aggresss/article/details/54928181>

# 第一期 前言 《虚拟机就是开发板》 - CSDN博客

来源网址: <https://blog.csdn.net/aggresss/article/details/54928181>

记得刚上大学时的第一年寒假，从图书馆借了10本厚厚的书带回了家，心想这个寒假一定要过得充实，把这几本书读透，结果不用想也知道，一本也没看懂。同样是大三的时候，一天室友从网上买了个51单片机的开发板，随后我把随机送的实验教程全都动手实践了一遍，这几个实验下来那叫一个通透，瞬间把《微机原理》《通信原理》这些理论课串联了起来。从此我发现了实践的重要性，一本教材，一个理论学习的正确打开方式应该是每一小步理论学习都伴随有相应的实践验证和探索。

学习是一件高风险的事情。首先方法不对根本学不明白，那样等于浪费时间；其次有些理论学习的实践环境在学习者所处的环境中根本不具备，没有实践的理论学习也是浪费时间；最后，学习只是能够产出结果的必要非充分条件，能够产出结果的必要非充分条件还有学习者的心态、知识基础、价值取向、运气等诸多因素。

不多废话了，来说主题，开发板可能对于电子类理工科人并不陌生，很多项目开发都需要开发板作为验证载体。对于接口类的验证确实需要物理开发板来验证，但是对于一些偏软件类的实践验证可以通过虚拟机或者叫模拟器来完成，这样可以避免物理平台的故障风险，降低实践成本，把更多的注意力放在需要关注的软件验证上面。这一次，我们来把虚拟机当做开发板，探讨通过虚拟机学习Linux相关知识的方法。

虚拟机的运行软件是QEMU。QEMU官方 ([www.qemu-project.org](http://www.qemu-project.org)) 对自己的定义为“QEMU is a generic and open source machine emulator and virtualizer.”名称取自“Quick EMUlator”。

QEMU支持3种运行模式：

Full-system emulation: Run operating systems for any machine, on any supported architecture

User-mode emulation: Run programs for another Linux/BSD target, on any supported architecture

Virtualization: Run KVM and Xen virtual machines with near native performance

其中，用户模式仿真允许一个 CPU 构建的进程在另一个 CPU 上执行（执行主机 CPU 指令的动态翻译并相应地转换 Linux 系统调用）；系统模式仿真允许对整个系统进行仿真，包括处理器和配套的外围设备；虚拟化运行模式是指通过底层的硬件支持（Intel-VT，AMD-V等虚拟化技术）和Xen或者KVM等系统级虚拟化技术相结合，高效运行虚拟化系统。

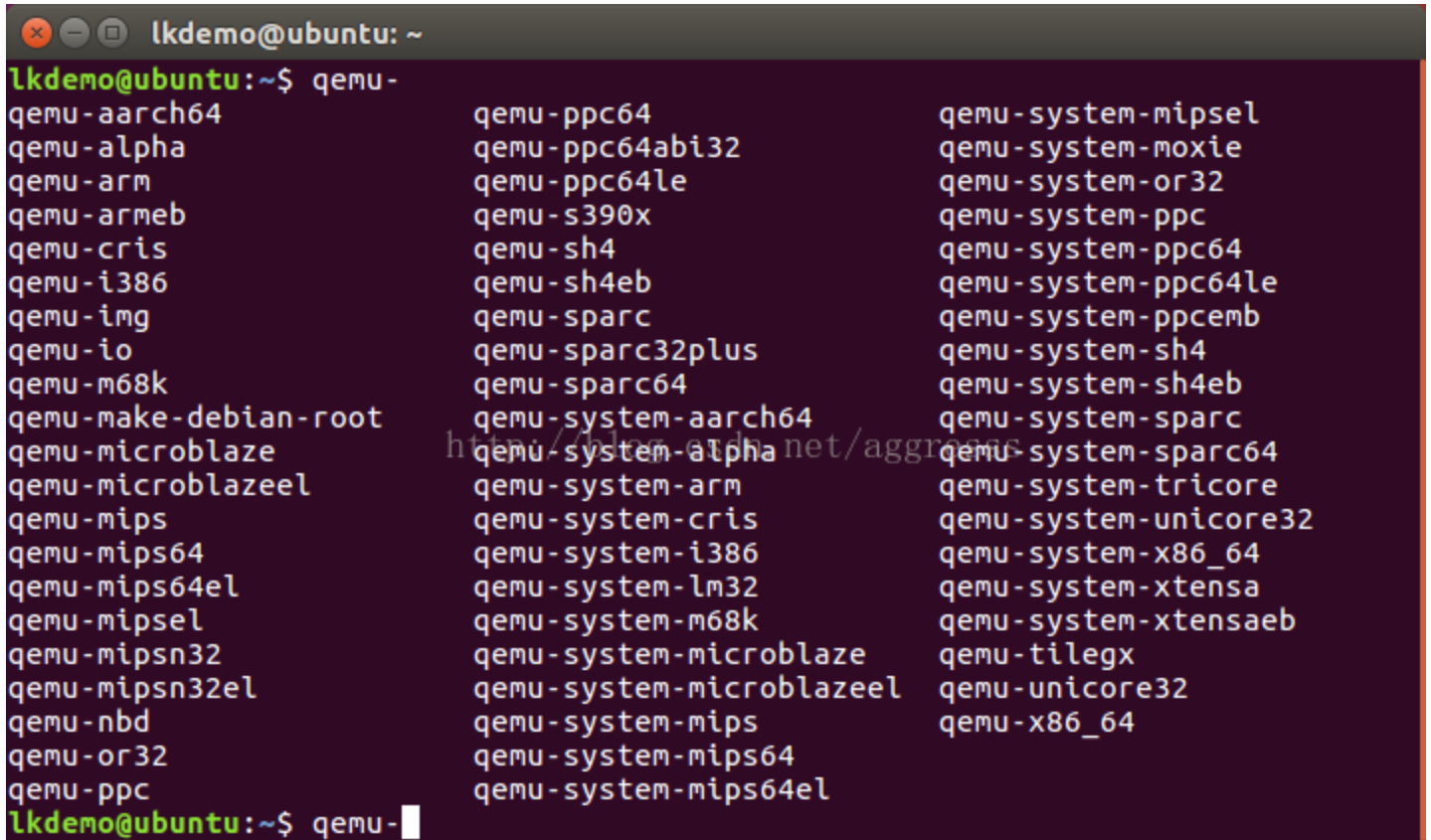
这里要分清两个概念，模拟器(emulation)和虚拟化(virtualization)，模拟器是emulator，通常是为了模拟不同指令集、不同体系架构的 CPU，所以多数情况要对微指令进行解释执行，效率相对与虚拟化慢很多；虚拟化技术 virtualization，基本都是去模拟一套相同指令集相同架构的硬件平台，因此在做好保护的前提下，很多时候可以直接利用 CPU 去执行目标指令。虽然还是模拟物理 CPU 而不借助于 Host OS 的功能，少了一层指令集转换，运行速度会提高不少。

对于使用过Vmware Workstation 或者是 Virtual Box 的人刚接触QEMU时比较不习惯，因为它是一个命令行下纯脚本控制的软件，这就是开源社区的风格，专注于做好自己的事，UI的事情会由另一个社区来处理（可以通过额外安装 qemu-launcher 实现）。

运行QEMU最好是在Linux环境下，我的运行环境是Ubuntu16.04\_amd64 版本。QEMU的安装非常简单，在ubuntu环境下输入：

```
sudo apt-get install qemu
```

等待安装完成后在shell中输入 `qemu` 然后按两下Tab键，就可以看到QEMU支持的所有命令，如下图所示。



```
lkdemo@ubuntu: ~  
lkdemo@ubuntu:~$ qemu-  
qemu-aarch64          qemu-ppc64          qemu-system-mipsel  
qemu-alpha           qemu-ppc64abi32     qemu-system-moxie  
qemu-arm             qemu-ppc64le        qemu-system-or32  
qemu-armeb           qemu-s390x           qemu-system-ppc  
qemu-cris             qemu-sh4             qemu-system-ppc64  
qemu-i386             qemu-sh4eb           qemu-system-ppc64le  
qemu-img             qemu-sparc           qemu-system-ppcemb  
qemu-io              qemu-sparc32plus     qemu-system-sh4  
qemu-m68k             qemu-sparc64         qemu-system-sh4eb  
qemu-make-debian-root qemu-system-aarch64  qemu-system-sparc  
qemu-microblaze       qemu-system-alpha    qemu-system-sparc64  
qemu-microblazeel     qemu-system-arm      qemu-system-tricore  
qemu-mips             qemu-system-cris     qemu-system-unicore32  
qemu-mips64           qemu-system-i386     qemu-system-x86_64  
qemu-mips64el         qemu-system-lm32     qemu-system-xtensa  
qemu-mipsel           qemu-system-m68k     qemu-system-xtensaeb  
qemu-mipsn32          qemu-system-microblaze qemu-tilegx  
qemu-mipsn32el        qemu-system-microblazeel qemu-unicore32  
qemu-nbd              qemu-system-mips     qemu-x86_64  
qemu-or32             qemu-system-mips64  
qemu-ppc              qemu-system-mips64el  
lkdemo@ubuntu:~$ qemu-
```

命令的最后一部分代表模拟的平台，中间带system的是运行在系统模式下，不带system的是运行在用户模式下。

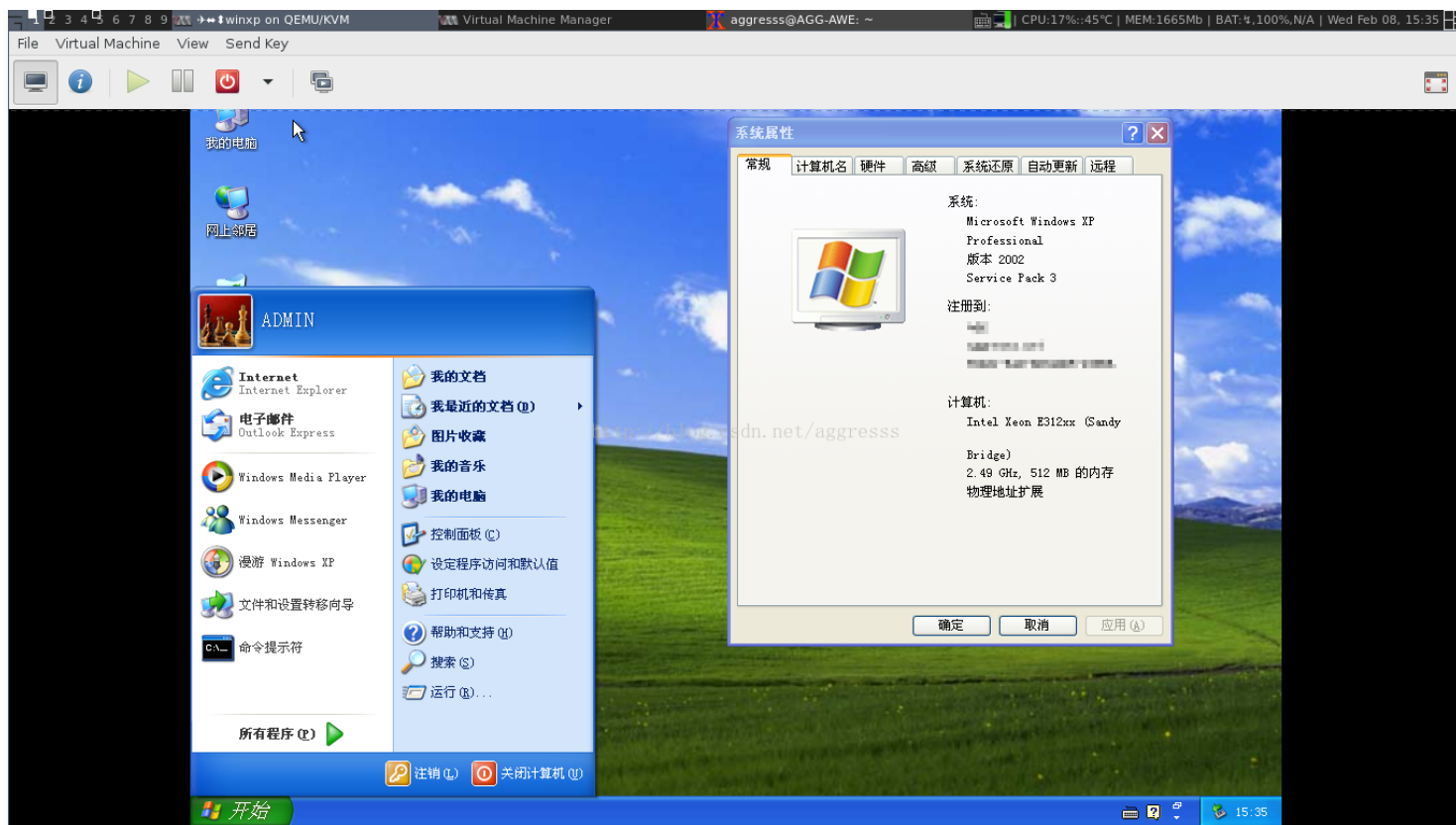
在接下来几期的实践中我们会使用 `qemu-system-arm` 模拟arm平台下的一个开发板，来进行Linux相关的一些实验。在模拟arm平台前，我们先体验一下QEMU与KVM相结合的虚拟化技术，来虚拟化运行一个Windows XP系统，体验一下QEMU可以实现Vmware Workstation 和 Virtual Box 同样的功能。

首先需要安装两个软件：

```
sudo apt-get install qemu-kvm  
sudo apt-get install virt-manager
```

其中，`qemu-kvm`提供虚拟化加速功能，`virt-manager`是一个轻量级应用程序套件，提供管理虚拟机的命令行或图形用户界面。

安装完成后通过命令 `virt-manager` 就可以启动图形化的虚拟机管理界面，准备好一张ISO格式的系统光盘，我这里用Windows XP，当然，你也可以选择任意支持光盘安装的系统ISO文件。各种下一步之后就可以在Linux环境下虚拟化运行Windows XP 系统了，体验了一下，相当流畅，毕竟XP是15年前的操作系统了，对硬件要求还是比较低的，下面是运行的截图。



系统运行后，使用 `ps -ef |grep qemu` 命令查看一下就会发现，其实virt-manager 调用了 libvirt+ 进程，然后 libvirt+ 通过调用 `qemu-system-x86_64` 命令来运行虚拟机，只是和模拟器不同的是它使用了 `-enable-kvm` 参数，就变成了虚拟化运行了。我们所看到的图形化的虚拟机最终都是通过调用命令行命令加参数实现的，这就是Linux社区软件的风格。

```
aggresss@AGG-AWE:~$ ps -ef |grep qemu
libvirt+ 1913      1 99 15:47 ?          00:00:19 qemu-system-x86_64 -enable-kvm -name
winnxp -S -machine pc-i440fx-xenial,accel=kvm,usb=off -cpu SandyBridge,hv_time,hv_re
laxed,hv_vapic,hv_spinlocks=0x1fff -m 512 -realtime mlock=off -smp 2,sockets=2,cores
=1,threads=1 -uuid 6e1968b6-a0e3-4bb0-bb2e-5a7d4641f3ab -no-user-config -nodefaults
-chardev socket,id=charmonitor,path=/var/lib/libvirt/qemu/domain-winnxp/monitor.sock,
server,nowait -mon chardev=charmonitor,id=monitor,mode=control -rtc base=localtime,d
riftfix=slew -global kvm-pit.lost_tick_policy=discard -no-hpet -no-shutdown -global
PIIX4.PM.disable_s3=1 -global PIIX4.PM.disable_s4=1 -boot strict=on -device ich9-usb
-ehci1,id=usb,bus=pci.0,addr=0x6.0x7 -device ich9-usb-uhci1,masterbus=usb.0,firstpor
t=0,bus=pci.0,multifunction=on,addr=0x6 -device ich9-usb-uhci2,masterbus=usb.0,first
port=2,bus=pci.0,addr=0x6.0x1 -device ich9-usb-uhci3,masterbus=usb.0,firstport=4,bus
=pci.0,addr=0x6.0x2 -device virtio-serial-pci,id=virtio-serial0,bus=pci.0,addr=0x5 -
drive file=/var/lib/libvirt/images/winnxp.qcow2,format=qcow2,if=none,id=drive-ide0-0-
0 -device ide-hd,bus=ide.0,unit=0,drive=drive-ide0-0-0,id=ide0-0-0,bootindex=1 -driv
e file=/home/aggresss/dpan/temp/zh-hans_windows_xp_professional_with_service_pack_3_
x86_cd_vl_x14-74070.iso,format=raw,if=none,id=drive-ide0-0-1,readonly=on -device ide
-cd,bus=ide.0,unit=1,drive=drive-ide0-0-1,id=ide0-0-1 -netdev tap,fd=26,id=hostnet0
-device rtl8139,netdev=hostnet0,id=net0,mac=52:54:00:df:42:82,bus=pci.0,addr=0x3 -ch
ardev pty,id=charserial0 -device isa-serial,chardev=charserial0,id=serial0 -chardev
spicevmc,id=charchannel0,name=vdagent -device virtserialport,bus=virtio-serial0.0,nr
=1,chardev=charchannel0,id=channel0,name=com.redhat.spice.0 -device usb-tablet,id=in
put0 -spice port=5900,addr=127.0.0.1,disable-ticketing,image-compression=off,seamles
s-migration=on -device qxl-vga,id=video0,ram_size=67108864,vram_size=67108864,vgamem
_mb=16,bus=pci.0,addr=0x2 -device intel-hda,id=sound0,bus=pci.0,addr=0x4 -device hda
-duplex,id=sound0-codec0,bus=sound0.0,cad=0 -chardev spicevmc,id=charredir0,name=usb
redir -device usb-redir,chardev=charredir0,id=redir0 -chardev spicevmc,id=charredir1
,name=usbredir -device usb-redir,chardev=charredir1,id=redir1 -device virtio-balloon
-pci,id=balloon0,bus=pci.0,addr=0x7 -msg timestamp=on
aggresss 2083  2044  0 15:47 pts/2    00:00:00 grep --color=auto qemu
aggresss@AGG-AWE:~$
```

通过上面这个简单的小实验，我们大体对QEMU有了一个感性的认识，从下一期开始，我们将使用QEMU模拟运行一个开发板，并在上面实验与Linux相关的内容。

《虚拟机就是开发板》英文名称是 Linux Kernel Demo ， 简称为 LKDemo 。

文章中提到的所有 开源文档、代码都提供下载：<https://github.com/aggresss/LKDemo>

## 第二期 QEMU模拟vexpress-a9开发板 《虚拟机就是开发板》 - CSDN博客

来源网址：<https://blog.csdn.net/aggresss/article/details/54942848>

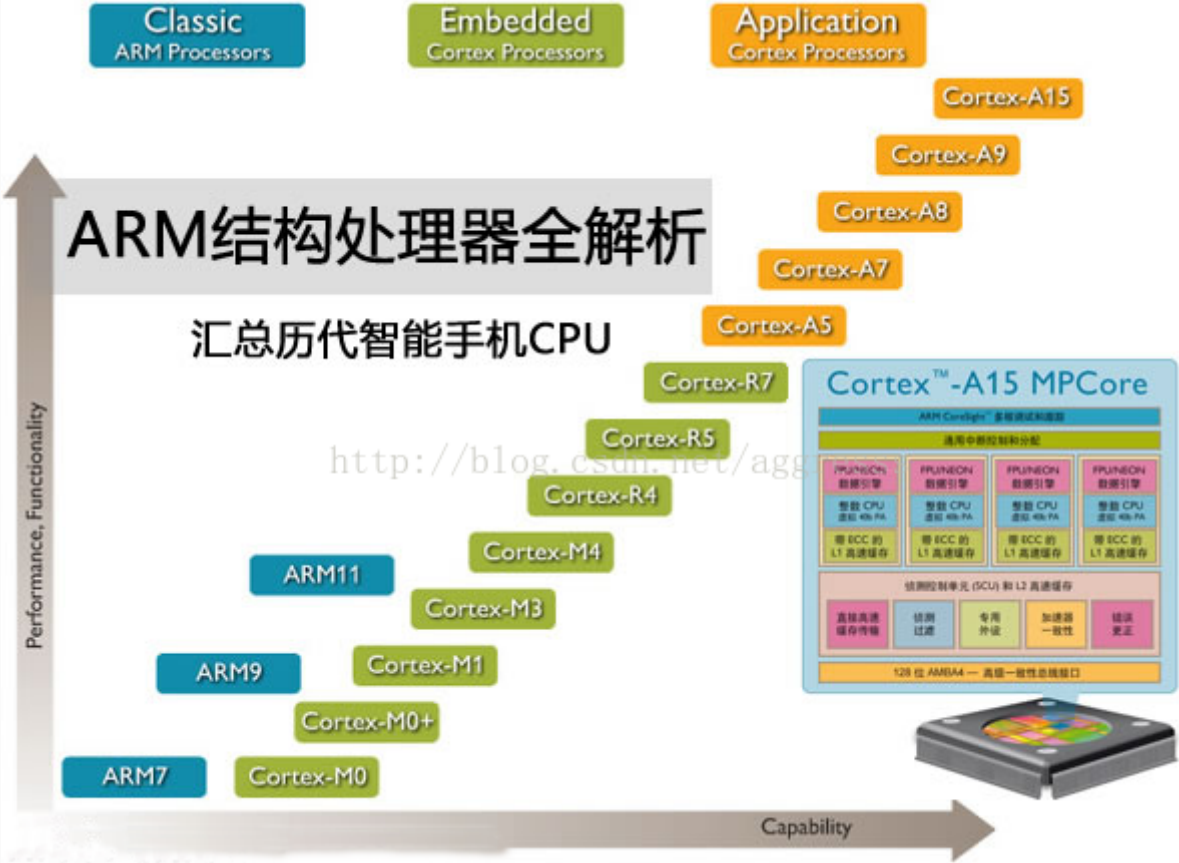
第一次接触的开发板是Intel的8051，自己买个DIP封装的芯片，加个晶振和复位就是个最小系统，非常适合入门。然后是STM32系列的开发板，运行一些RTOS系统，非常适合做一些小玩具。再往后接触的开发板就是ARM架构的了，国内的ARM开发板多数都被Samsung的芯片给占了，TI、NXP的芯片相比之下都太小众了，不容易形成大的社区群体；三星的ARM芯片用于开发板的比较大众的有：

|       |         |         |         |           |            |
|-------|---------|---------|---------|-----------|------------|
| ARM架构 | ARM7    | ARM9    | ARM11   | Cortex-A8 | Cortex-A9  |
| 对应芯片  | S3C44B0 | S3C2440 | S3C6410 | S5PV210   | Exynos4412 |



如果从资料的完整度和用户数量来看，S3C2440肯定是排在第一位的，当然对于现在Android火热的环境下，S3C6410作为可以最低运行Android系统的开发板也还有市场，目前Exynos4412是开发板市场主流。

下图是ARM处理器架构发展的天梯图，由于图片的年代比较久远，没有标出A53、A72等架构，仅供参考。



前面说了我对市面上流行的开发板的一些认识，都是物理意义上的开发板，而我们这一期要操作的是使用QEMU模拟一个ARM开发板。

在系统环境下输入：`qemu-system-arm -M help` 可以查看QEMU支持的ARM平台的开发板的型号，如下图所示。

```

aggresss@AGG-AWE:~$ qemu-system-arm -M help
Supported machines are:
akita           Sharp SL-C1000 (Akita) PDA (PXA270)
borzoi          Sharp SL-C3100 (Borzoi) PDA (PXA270)
canon-all00    Canon PowerShot A1100 IS
cheetah         Palm Tungsten|E aka. Cheetah PDA (OMAP310)
collie          Sharp SL-5500 (Collie) PDA (SA-1110)
connex          Gumstix Connex (PXA255)
cubieboard      cubietech cubieboard
highbank        Calxeda Highbank (ECX-1000)
imx25-pdk       ARM i.MX25 PDK board (ARM926)
integratorcp    ARM Integrator/CP (ARM926EJ-S)
kzm             ARM KZM Emulation Baseboard (ARM1136)
lm3s6965evb     Stellaris LM3S6965EVB
lm3s811evb      Stellaris LM3S811EVB
mainstone       Mainstone II (PXA27x)
midway          Calxeda Midway (ECX-2000)
musicpal        Marvell 88w8618 / MusicPal (ARM926EJ-S)
n800            Nokia N800 tablet aka. RX-34 (OMAP2420)
n810            Nokia N810 tablet aka. RX-44 (OMAP2420)
netduino2       Netduino 2 Machine
none            empty machine
nuri            Samsung NURI board (Exynos4210)
realview-eb     ARM RealView Emulation Baseboard (ARM926EJ-S)
realview-eb-mpcore ARM RealView Emulation Baseboard (ARM11MPCore)
realview-pb-a8  ARM RealView Platform Baseboard for Cortex-A8
realview-pbx-a9 ARM RealView Platform Baseboard Explore for Cortex-A9
smdkc210        Samsung SMDKC210 board (Exynos4210)
spitz           Sharp SL-C3000 (Spitz) PDA (PXA270)
sx1             Siemens SX1 (OMAP310) V2
sx1-v1          Siemens SX1 (OMAP310) V1
terrier         Sharp SL-C3200 (Terrier) PDA (PXA270)
tosa            Sharp SL-6000 (Tosa) PDA (PXA255)
verdex          Gumstix Verdex (PXA270)
versatileab     ARM Versatile/AB (ARM926EJ-S)
versatilepb     ARM Versatile/PB (ARM926EJ-S)
vexpress-a15    ARM Versatile Express for Cortex-A15
vexpress-a9     ARM Versatile Express for Cortex-A9
virt            ARM Virtual Machine
xilinx-zynq-a9  Xilinx Zynq Platform Baseboard for Cortex-A9
z2              Zipit Z2 (PXA27x)
aggresss@AGG-AWE:~$

```

结合一下网上的资料后发现 有关 vexpress-a9 的资料和讨论最多，所以我们选择这个开发板来进行模拟。

vexpress系列（全称Versatile Express Family）是ARM自己推出的开发板，主要是方便SOC厂商设计、验证和测试自己的SOC芯片设计用的，官方的解释是：

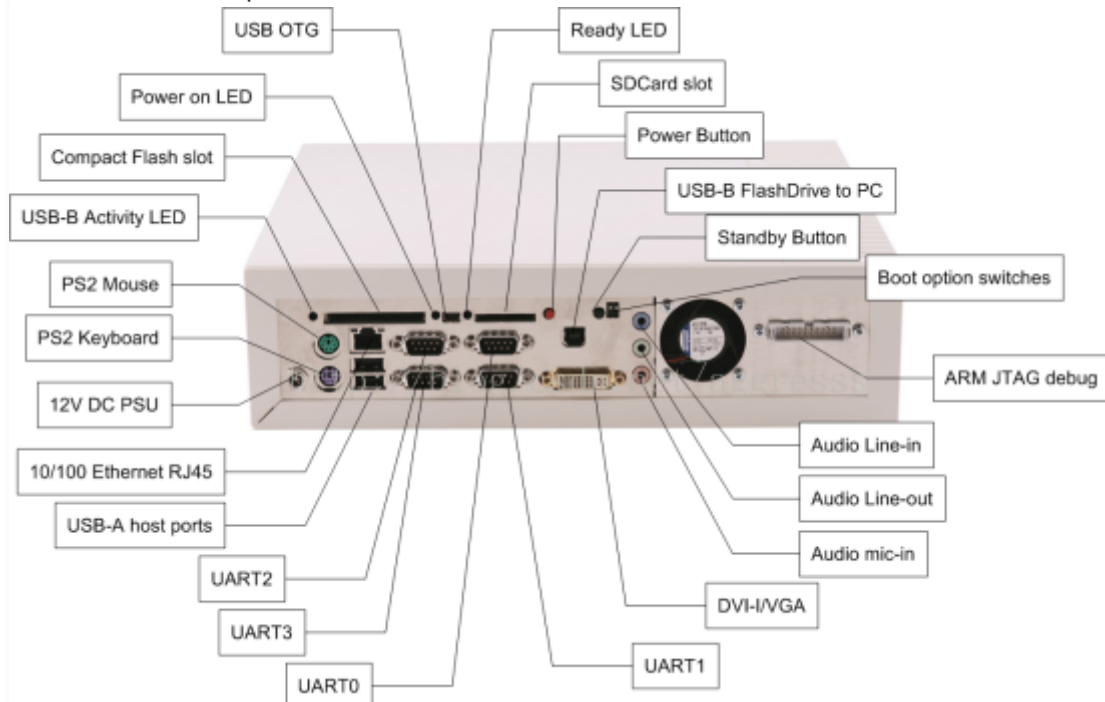
ARM® Versatile™ Express development boards are the ideal platform for accelerating the development and reducing the risk of new SoC designs. The combination of ASIC and FPGA technology in ARM boards delivers an optimal solution in terms of speed, accuracy, flexibility and cost.

更多相关内容请参考：<https://developer.arm.com/products/system-design/versatile-express-family>

虽然我们要用软件模拟运行这个开发板，但硬件结构还是要了解的，这样才能做到知己知彼。由于vexpress的面向用户是SOC设计者，所以设计方法也很另类，采用了主板+子板的设计结构，主板提供各种外围接口，子板提供CPU运算。

主板 Express uATX（或 V2M-P1）是 Versatile Express 系列中的第一款可用主板。它嵌在 uATX 大小的两件式漂亮塑料外壳内。活动式壳盖可以露出主板以便安装子板和连接测试设备。所有连接器和控制器都安装在后面板上。此主

板有两组子板牛角连接器。它必须始终与处理器子板 Express 或软宏模型子板配对使用以提供主系统处理器。可以添加可选的逻辑子板 Express 板以提供自定义 IP 开发和验证功能。



处理器子板 Express 板在 Versatile™ Express 开发系统中提供主系统 CPU。处理器子板 必须与提供电源、配置和外设连接的主板 Express uATX 板配对使用。处理器子板 Express 板与 Versatile 产品系列中的前代产品的不同之处在于，其内存和 LCD 控制器等高带宽外设是与 ARM 处理器一起在测试芯片中实现的。这会显示提升性能，使系统更适合进行软件基准测试并完全能运行 Debian Linux 等桌面操作系统。



更多的详细内容请参考：

主板：

<https://www.arm.com/zh/products/tools/development-boards/versatile-express/motherboard-express.php>

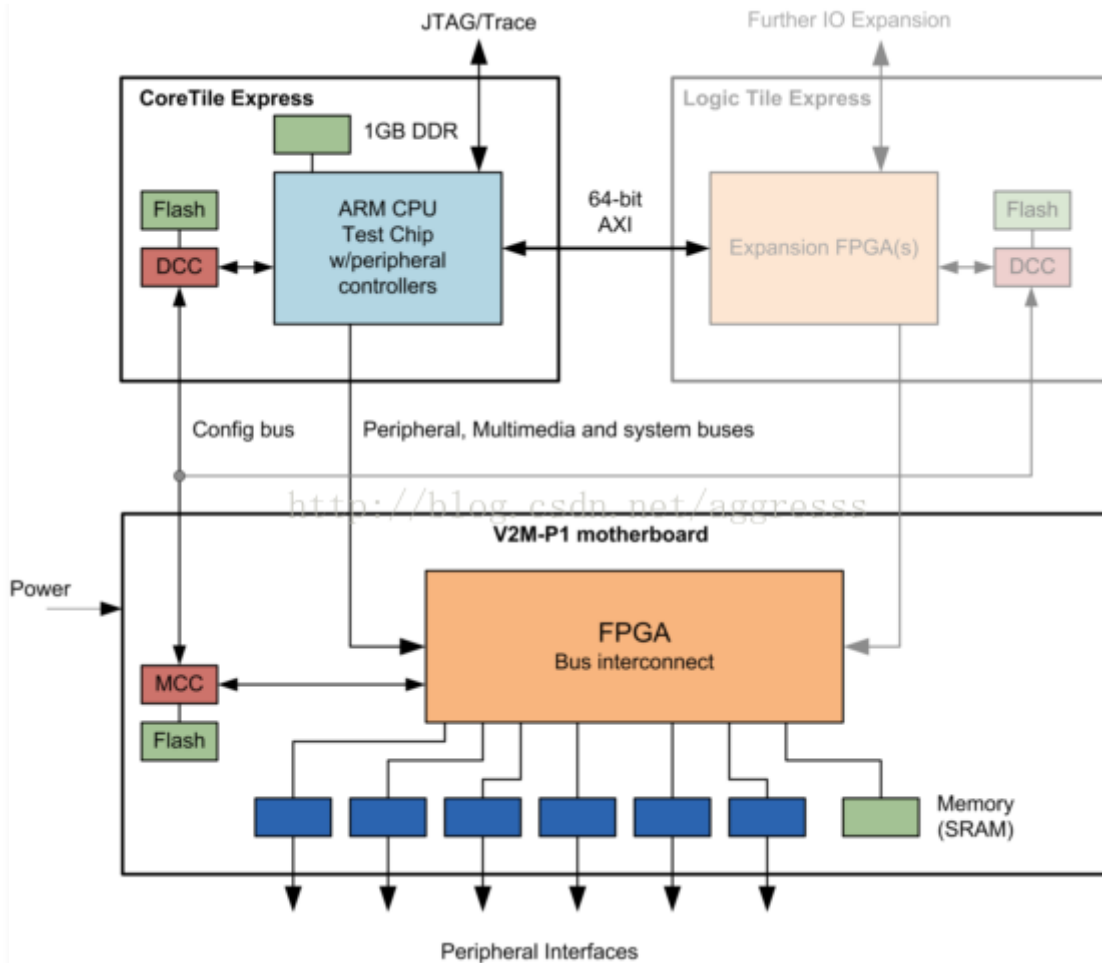
处理器子板：

<https://www.arm.com/zh/products/tools/development-boards/versatile-express/coretile-express.php>

文档下载：

<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.boards/index.html>

系统原理图：



虽然看上去比较复杂，但是如果我们软件模拟的话就把它当成一个整体就是了。运行系统后和正常的开发板没什么区别。下面我们来实际动手用QEMU模拟运行一下vexpress-a9开发板。

由于我们还没有编译和开发板相关的程序，所以这里先使用一个我编译好的U-Boot程序作为载体运行一下看看效果，先对模拟器模拟开发板有一个感性的认识，安装好qemu软件后，在github上的test目录下<https://github.com/aggress/LKDemo> 有u-boot.sh和u-boot两个文件。将这两个文件放到同一个目录下然后运行 `./u-boot.sh` 就可以启动开发板了，下图是运行后的输出信息。



```
aggresss@AGG-AWE:~$ cd workspace/LKDemo/test/
aggresss@AGG-AWE:~/workspace/LKDemo/test$ ./u-boot.sh
pulseaudio: set_sink_input_volume() failed
pulseaudio: Reason: Invalid argument
pulseaudio: set_sink_input_mute() failed
pulseaudio: Reason: Invalid argument

U-Boot 2017.01 (Feb 08 2017 - 20:27:04 +0800)

DRAM: 512 MiB
WARNING: Caches not enabled
Flash: 128 MiB
MMC: MMC: 0
*** Warning - bad CRC, using default environment

In: serial
Out: serial
Err: serial
Net: smc911x-0
Hit any key to stop autoboot: 0
=> version

U-Boot 2017.01 (Feb 08 2017 - 20:27:04 +0800)
arm-linux-gnueabi-gcc (Ubuntu/Linaro 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609
GNU ld (GNU Binutils for Ubuntu) 2.26.1
=> █
```

QEMU的相关参数说明请参考: <https://qemu.weilnetz.de/doc/qemu-doc.html>

## 第三期 QEMU调试U-Boot实验 《虚拟机就是开发板》 - CSDN博客

来源网址: <https://blog.csdn.net/aggresss/article/details/54945726>

这一期注定会很简短, 简短的意义就在于使用模拟器做某些事情确实很快捷。这一期我们使用QEMU来模拟运行U-Boot, 大家对U-Boot应该都不陌生, 相当于Linux的学前班, U-Boot的数据结构定义和驱动模型定义都采用Linux风格, 在研究Linux内核前分析一下U-Boot会大有裨益。

记得第一次用物理开发板调试U-Boot时, 怎么把U-Boot下载到开发板就让我研究了一星期, 把有限的精力都浪费在了物理性的问题上了, 而对于U-Boot的内部软件架构就会分散一些精力。下面我们来实践一下在模拟器上运行U-Boot, 如果你之前没有编译过U-Boot, 没有把它下载到物理开发板并运行过, 都不要紧, 只要按照下面的操作来, 20分钟走完全程。

1. 首先安装交叉编译器, 执行: `sudo apt-get install gcc-arm-linux-gnueabi`
2. 下载U-Boot源文件: <http://ftp.denx.de/pub/u-boot/> 我下载的是 u-boot-2017.01.tar.bz2
3. 解压源文件 `tar jvxf u-boot-2017.01.tar.bz2 -C xxx` (xxx为需要解压的目录)
4. 进入U-Boot 源文件目录, 然后执行:  
`export ARCH=arm`  
`export CROSS_COMPILE=arm-linux-gnueabi-`  
`make vexpress_ca9x4_defconfig`  
`make`

编译完成后，如果目录下生成 u-boot 文件，则说明编译成功。

#### 5. 在U-Boot源码目录下编写脚本 run.sh

```
qemu-system-arm \  
-M vexpress-a9 \  
-nographic \  
-m 512M \  
-kernel u-boot
```

然后 `chmod +x run.sh` 增加文件执行权限。

#### 6. 最后执行 `./run.sh`

```
aggresss@AGG-AWE:~$ cd workspace/LKDemo/test/  
aggresss@AGG-AWE:~/workspace/LKDemo/test$ ./u-boot.sh  
pulseaudio: set_sink_input_volume() failed  
pulseaudio: Reason: Invalid argument  
pulseaudio: set_sink_input_mute() failed  
pulseaudio: Reason: Invalid argument  
  
U-Boot 2017.01 (Feb 08 2017 - 20:27:04 +0800)  
  
DRAM: 512 MiB  
WARNING: Caches not enabled  
Flash: 128 MiB  
MMC: MMC: 0  
*** Warning - bad CRC, using default environment  
  
In: serial  
Out: serial  
Err: serial  
Net: smc911x-0  
Hit any key to stop autoboot: 0  
=> version  
  
U-Boot 2017.01 (Feb 08 2017 - 20:27:04 +0800)  
arm-linux-gnueabi-gcc (Ubuntu/Linaro 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609  
GNU ld (GNU Binutils for Ubuntu) 2.26.1  
=>
```

到这里，你已经搭建好了一个U-Boot的调试环境。如果你想研究一下U-Boot，这是一个非常好的开始，没必要被那些物理性的各种未知问题所困扰。

理解U-Boot内部运行原理的一个捷径就是对它进行修改，然后验证性的调试运行，这样能为你揭示出仅仅通过看代码无法看到的深层机理。

上面这些步骤是不是很快就完成了，我们用最短的时间制作出了一个U-Boot的调试验证环境，接下来怎么去分析和研究U-Boot就看你的啦。

## 第四期 QEMU调试Linux内核实验 《虚拟机就是开发板》 - CSDN博客

来源网址: <https://blog.csdn.net/aggresss/article/details/54946438>

这一期我们来制作一个Linux的最小系统，让它在虚拟开发板上运行。整体的流程类似于LFS (<http://www.linuxfromscratch.org/>)，不过LFS的目标是制作一个功能较完善的Linux发行版，而我们要做的是最小系统，步骤会精简很多，大体就分成三个步骤：1.编译内核；2.制作根文件系统；3.调试运行。

## 第一步：编译内核

1. 首先安装交叉编译器，执行： `sudo apt-get install gcc-arm-linux-gnueabi`  
2. 下载内核源文件 <https://www.kernel.org/pub/linux/kernel/>，我这里下载的是 4.1.38 版本，一个比较稳定的版本；

3. 解压内核 `tar zxvf linux-4.1.38.tar.gz -C xxxx`（xxxx为需要解压的目录）  
4. 进入kernel 源文件目录，然后执行：

```
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabi-
make vexpress_defconfig
make zImage
make modules
make dtbs
```

编译后生成在 arch/arm/boot 目录下生成 zImage 文件，则说明编译成功。

## 第二步：制作根文件系统

1. 制作根文件系统首先要生成一个虚拟磁盘，创建一个虚拟磁盘的两种方法：

```
dd if=/dev/zero of=vexpress.img bs=512 count=$((2*1024*100))
qemu-img create -f raw vexpress.img 100M
```

这两种方法任选一种执行就可以，目的就是生成一个vexpress.img的虚拟镜像文件，为了更好的兼容性选择 raw 格式的镜像。

2. 虚拟磁盘中创建分区并修改：

- (1). `fdisk vexpress.img`，然后使用 `n` 命令创建分区，各种下一步就行；
- (2). `losetup /dev/loop0 vexpress.img`，挂载vexpress.img到/dev/loop0设备上；
- (3). `partx -u /dev/loop0`，使用partx命令让系统刷新系统的分区信息；
- (4). `mkfs.ext2 /dev/loop0p1`，制作ext2格式的文件系统；
- (5). `mkdir rootfs`，建立一个rootfs目录用来作为挂载目录；

`mount -o loop /dev/loop0p1 ./rootfs`，将生成的ext2格式的分区挂载到rootfs目录；

(6). 执行到这里虚拟磁盘就已经制作好了，下面的两个步骤是卸载磁盘时的操作，可以先跳过，直接到第三节去编译 busybox；

- (7). `partx -d /dev/loop0`，卸载loop0设备下的分区；如果执行不成功可以试试 `sudo umount -f rootfs`
- (8). `losetup -d /dev/loop0`，卸载loop0设备；

说明：

如果直接使用 `mkfs.ext3` 写入img文件，无法使用fdisk显示分区信息；

`fdisk n`命令 默认的first sector 是2048扇区；默认分区名为 文件名+分区序号；

`mount` 挂载空分区会报错 `wrong fs type, bad option, bad superblock`；

`losetup /dev/loop0 vexpress.img` 命令相当于mount命令中的 `-o loop` 参数；

`partx -u /dev/loop0` 强制内核刷新可识别分区；

查看分区类型 `df -Th`；

3. 编译 Busybox：

- (1). 下载Busybox <https://busybox.net/downloads/> , 我下载的版本是 1.26.2 , 一个比较稳定的版本;
- (2). 解压内核 tar jvxf busybox-1.26.2.tar.bz2 -C xxxx (xxx为需要解压的目录)
- (3). 进入 Busybox 源文件目录下执行 make menuconfig  
做如下配置:

```
1 | Busybox Settings --->
2 | Build Options --->
3 | [*] Build BusyBox as a static binary (no shared libs)
4 | (arm-linux-gnueabi-) Cross Compiler prefix
```

使用交叉编译器编译Busybox

- (4). 执行 make 编译Busybox
- (5). 执行 make install 会在 \_install 目录下生成 需要的文件 bin linuxrc sbin usr ;

#### 4.制作根文件系统:

- (1). 拷贝busybox

```
sudo cp -raf busybox/_install/* rootfs/
```

- (2). 拷贝运行库

```
sudo cp -arf /usr/arm-linux-gnueabi/lib rootfs/
sudo rm rootfs/lib/*.a
sudo arm-linux-gnueabi-strip rootfs/lib/*
```

- (3). 创建必要目录

```
sudo mkdir -p rootfs/proc/
sudo mkdir -p rootfs/sys/
sudo mkdir -p rootfs/tmp/
sudo mkdir -p rootfs/root/
sudo mkdir -p rootfs/var/
sudo mkdir -p rootfs/mnt/
```

- (4). 创建必要节点

```
sudo mkdir -p rootfs/dev/
sudo mknod rootfs/dev/tty1 c 4 1
sudo mknod rootfs/dev/tty2 c 4 2
sudo mknod rootfs/dev/tty3 c 4 3
sudo mknod rootfs/dev/tty4 c 4 4
sudo mknod rootfs/dev/console c 5 1
sudo mknod rootfs/dev/null c 1 3
```

- (5). 制作必要etc文件

etc 目录下的必要文件有5个: fstab, init.d/rcS, inittab, profile, sysconfig/HOSTNAME

我把这五个文件放到 <https://github.com/aggresss/LKDemo> 中 tools 目录下的 etc.tar.gz 里, 可以下载并

解压后使用;

```
sudo cp -arf etc rootfs/
```

到这里根文件系统就已制作完成, 退出rootfs目录并执行 losetup -d /dev/loop0 卸载虚拟磁盘文件。

### 第三步: 调试运行



1. 启动Linux最小系统需要我们刚才生成的3个文件：

- (1). Linux kernel目录下 arch/arm/boot/dts/vexpress-v2p-ca9.dtb 文件;
- (2). Linux kernel目录下 arch/arm/boot/zImage 文件
- (3). 生成的虚拟磁盘文件: vexpress.img;

将这三个文件放到同一个目录下。

2. 制作启动脚本：

在上面三个文件的同目录下创建启动脚本 vim run\_linux.sh

```
1 | qemu-system-arm \  
2 | -nographic \  
3 | -sd vexpress.img \  
4 | -M vexpress-a9 \  
5 | -m 512M \  
6 | -kernel zImage \  
7 | -dtb vexpress-v2p-ca9.dtb \  
8 | -smp 4 \  
9 | -append "init=/linuxrc root=/dev/mmcblk0p1 rw rootwait earlyprintk console=ttyAMA0"
```

增加可执行权限 chmod +x run\_linux.sh

3. 调试运行：

执行 ./run\_linux.sh，即可在模拟的开发板上运行Linux系统，下面是运行后的截图：

```

1 2 3 4 5 6 7 8 9 X aggresss@AGG-AWE: ~/workspace/LKDemo/qemu.d X aggresss@AGG-AWE: ~
smc911x 4e000000.ethernet eth0: MAC Address: 52:54:00:12:34:56
isp1760 4f000000.usb: bus width: 32, oc: digital
isp1760 4f000000.usb: NXP ISP1760 USB Host Controller
isp1760 4f000000.usb: new USB bus registered, assigned bus number 1
isp1760 4f000000.usb: Scratch test failed.
isp1760 4f000000.usb: can't setup: -19
isp1760 4f000000.usb: USB bus 1 deregistered
usbcore: registered new interface driver usb-storage
mousedev: PS/2 mouse device common for all mice
rtc-pl031 10017000.rtc: rtc core: registered pl031 as rtc0
mmci-pl18x 10005000.mmci: Got CD GPIO
mmci-pl18x 10005000.mmci: Got WP GPIO
mmci-pl18x 10005000.mmci: No vqmmc regulator found
mmci-pl18x 10005000.mmci: mmc0: PL181 manf 41 rev0 at 0x10005000 irq 34,35 (pio)
ledtrig-cpu: registered to indicate activity on CPUs
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
aaci-pl041 10004000.aaci: ARM AC'97 Interface PL041 rev0 at 0x10004000, irq 33
aaci-pl041 10004000.aaci: FIFO 512 entries
oprofile: using arm/armv7-ca9
NET: Registered protocol family 17
9pnet: Installing 9P2000 support
Registering SWP/SWPB emulation handler
input: AT Raw Set 2 keyboard as /devices/platform/smb/smb:motherboard/smb:motherboard:iofpga@
7,00000000/10006000.kmi/serio0/input/input0
rtc-pl031 10017000.rtc: setting system clock to 2017-02-09 03:42:39 UTC (1486611759)
ALSA device list:
 #0: ARM AC'97 Interface PL041 rev0 at 0x10004000, irq 33
mmc0: new SD card at address 4567
mmcblk0: mmc0:4567 QEMU! 100 MiB
 mmcblk0: p1
input: ImExPS/2 Generic Explorer Mouse as /devices/platform/smb/smb:motherboard/smb:motherboa
rd:iofpga@7,00000000/10007000.kmi/serio1/input/input2
EXT2-fs (mmcblk0p1): warning: mounting unchecked fs, running e2fsck is recommended
VFS: Mounted root (ext2 filesystem) on device 179:1.
Freeing unused kernel memory: 260K (8061d000 - 8065e000)
random: nonblocking pool is initialized

Please press Enter to activate this console.
[root@vexpress ]#
[root@vexpress ]# uname -a
Linux vexpress 4.1.38 #1 SMP Sat Jan 21 20:34:08 CST 2017 armv7l GNU/Linux
[root@vexpress ]#

```

#### 4. 关闭模拟开发板进程:

将下面的命令做成脚本运行便可以彻底关闭已经运行的QEMU进程:

```
ps -A | grep qemu-system-arm | awk '{print $1}' | xargs sudo kill -9
```

这一期我们来集中解决一个问题: 虚拟开发板与主机之间文件共享的方法。在上一期中我们将目标板运行了自己制作的最小Linux系统, 然而这个最小系统里面并没有编译环境, 也就是说如果我们想编译程序在目标开发板上运行需要先在本地主机上进行交叉编译, 然后将生成的二进制目标文件上传到目标开发板上运行, 那么问题来了, 我们需要一种快捷的文件共享方法。

使用物理开发板时最常用的方法是NFS, 所以我们也通过NFS来共享文件, 我们在主机上面建立NFS服务, 然后在目标开发板上进行挂载。想实现这个功能需要分成两步来进行: 1. 主机与目标开发板之间网络连通; 2. 主机建立NFS服务, 目标开发板挂载。

QEMU提供的4种通信方式与外界联网:

1. User mode stack: 这种方式在qemu进程中实现一个协议栈, 负责在虚拟机VLAN和外部网络之间转发数据。可以将该协议栈视为虚拟机与外部网络之间的一个NAT服务器, 外部网络不能主动与虚拟机通信。虚拟机VLAN中的各个网络接口只能置于10.0.2.0子网中, 所以这种方式只能与外部网络进行有限的通信。此外, 可以用-redir选项为宿主机和虚拟机的两个TCP或UDP端口建立映射, 实现宿主机和虚拟机在特殊要求下的通信(例如X-server或ssh)。User mode stack通信方式由-net user选项启用, 如果不显式指定通信方式, 则这种方式是qemu默认的通信方式。

2. socket: 这种方式又分为TCP和UDP两种类型。

(1). TCP: 为一个VLAN创建一个套接字, 让该套接字在指定的TCP端口上监听, 而其他VLAN连接到该套接字上, 从而将多个VLAN连接起来。缺点在于如果监听套接字所在qemu进程崩溃, 整个连接就无法工作。监听套接字所在VLAN通过-net socket,listen选项启用, 其他VLAN通过-net socket,connect选项启用。

(2). UDP: 所有VLAN连接到一个多播套接字上, 从而使多个VLAN通过一个总线通信。所有VLAN都通过-net socket,mcast选项启用。

3. TAP: 这种方式首先需要在宿主机中创建并配置一个TAP设备, qemu进程将该TAP设备连接到虚拟机VLAN中。其次, 为了实现虚拟机与外部网络的通信, 在宿主机中通常还要创建并配置一个网桥, 并将宿主机的网络接口(通常是eth0)作为该网桥的一个接口。最后, 只要将TAP设备作为网桥的另一个接口, 虚拟机VLAN通过TAP设备就可以与外部网络完全通信了。这是因为, 宿主机的eth0接口作为网桥的接口, 与外部网络连接; TAP设备作为网桥的另一个接口, 与虚拟机VLAN连接, 这样两个网络就连通了。此时, 网桥在这两个网络之间转发数据帧。

4. VDE: 这种方式首先要启动一个VDE进程, 该进程打开一个TAP设备, 然后各个虚拟机VLAN与VDE进程连接, 这样各个VLAN就可以通过TAP设备连接起来。VDE进程通过执行vde\_switch命令启动, 各个VLAN所在qemu进程通过执行veqe命令启动, 这些VLAN就可以与VDE进程连接了。

根据我们需要通过NFS通信的方式, 选择TAP的方式使主机和虚拟机进行通信。但是虚拟机只需要与主机互联, 不需要与外界互联, 所以并不需要创建网桥, 只要创建一个TAP就可以。

这里说一下Linux中tun/tap。tun/tap 驱动程序实现了虚拟网卡的功能, tun表示虚拟的是点对点设备, tap表示虚拟的是以太网设备, 这两种设备针对网络包实施不同的封装。利用tun/tap 驱动, 可以将tcp/ip协议栈处理好的网络分包传给任何一个使用tun/tap驱动的进程, 由进程重新处理后再发到物理链路中。

这是官方的解释:

TUN and TAP are virtual network kernel devices. Being network devices supported entirely in software, they differ from ordinary network devices which are backed up by hardware network adapters.

TUN (namely network TUNnel) simulates a network layer device and it operates with layer 3 packets like IP packets. TAP (namely network tap) simulates a link layer device and it operates with layer 2 packets like Ethernet frames. TUN is used with routing, while TAP is used for creating a network bridge.

Packets sent by an operating system via a TUN/TAP device are delivered to a user-space program which attaches itself to the device. A user-space program may also pass packets into a TUN/TAP device. In this case the TUN/TAP device delivers (or "injects") these packets to the operating-system network stack thus emulating their reception from an external source.

下面开始操作:

sudo apt-get install uml-utilities (User-Mode Linux, 使用它创建TAP)

sudo apt-get install bridge-utils (如果需要创建bridge可以选择安装)

安装完成后, 执行 `sudo tuncctl -u root -t tap30` 就可以在主机上创建一个网络设备, 这是使用 `ifconfig -a` 命令可以看到名字为tap30的网络设备。

执行 `ifconfig tap30 192.168.111.1 promisc up` 配置网卡IP地址, 并且以混杂模式启用。

主机的网络配置已经完成, 接下来开始配置虚拟机, QEMU通过 `-net` 参数指定网络配置

有三种选项

-net nic 必须有的基本配置,macaddr 设置mac地址, model是网卡的类型, 可以model=?查看有哪些类型

-net tap 使用桥接模式的, 需要指定启动脚本 `script=`和关闭脚本 `downscript`, fd是指向已经有的tap设备, name是在monitor模式使用info network看到的名字,ifname是tap设备在主机中的名字

-net user 用户模式, qemu使用Slirp实现了一整套tcp/ip协议栈

一般nic必须有, tap和user选一个使用

在上一期启动qemu的脚本中添加两行参数:

-net nic,vlan=0 \

-net tap,vlan=0,ifname=tap30,script=no,downscript=no

然后运行脚本启动虚拟机, 可能需要root权限才能启动。启动后在目标开发板中执行 `ifconfig -a` 就能看到 eth0 的网络接口, 默认都是没启用, 需要手动开启, 在模拟开发板中执行:

`ifconfig eth0 192.168.111.2 promisc up`

成功启动网卡后便可以使用 `ping 192.168.111.1` 来测试与主机的通信是否正常, 下面是运行截图:

```
Please press Enter to activate this console.
[root@vexpress]#
[root@vexpress]# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:56
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:31

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

[root@vexpress]# ifconfig eth0 192.168.111.2 promisc up
smsc911x 4e000000.ethernet eth0: SMSC911x/921x identified at 0xa0aa0000, IRQ: 31
device eth0 entered promiscuous mode
[root@vexpress]# ping 192.168.111.1
PING 192.168.111.1 (192.168.111.1): 56 data bytes
64 bytes from 192.168.111.1: seq=0 ttl=64 time=31.527 ms
64 bytes from 192.168.111.1: seq=1 ttl=64 time=1.958 ms
64 bytes from 192.168.111.1: seq=2 ttl=64 time=1.349 ms
64 bytes from 192.168.111.1: seq=3 ttl=64 time=1.370 ms
64 bytes from 192.168.111.1: seq=4 ttl=64 time=1.393 ms
^C
--- 192.168.111.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 1.349/7.519/31.527 ms
[root@vexpress]#
```

目标开发板与主机的通信问题解决后, 开始配置并启用NFS服务。在主机中输入:





成功添加Windows中的NFS客户端后，在Windows的CMD中输入showmount -e xxx.xxx.xxx.xxx（服务器IP地址）来查看可用的挂载目录。

然后用 mount IP:/挂载目录 盘符: 命令即可挂载完成NFS共享目录的挂载，下面是运行截图：

```
C:\Users\AGG>
C:\Users\AGG>showmount -e 192.168.2.8
导出列表在 192.168.2.8:
/mnt/dpan/share                192.168.111.2, 192.168.2.*

C:\Users\AGG>mount 192.168.2.8:/mnt/dpan/share Y:
Y: 现已成功连接到 192.168.2.8:/mnt/dpan/share
http://blog.csdn.net/aggresss
命令已成功完成。

C:\Users\AGG>Y:
Y:\>
```

## 第六期 基于QEMU进行Linux内核模块实验 《虚拟机就是开发板》 - CSDN博客

来源网址: <https://blog.csdn.net/aggresss/article/details/54956686>

对于Linux内核的学习，多数都是从调试运行内核模块开始的，这一期我们来总结一下用模拟开发板调试运行内核模块的一般方法。

首先写一个内核模块的helloworld源文件，包括hello.c 和相应的Makefile:

hello.c

```
1  /*
2   * A simple module for helloworld
3   *
4   * Copyright (C) 2017 aggresss (aggresss.163.com)
5   *
6   * Licensed under GPLv2 or later.
7   */
8
9
10 // Defining __KERNEL__ and MODULE allows us to access kernel-level code not usually available
11 #undef __KERNEL__ | #define __KERNEL__
12
13 #undef MODULE
14 #define MODULE
15
16 #include <linux/module.h> // included for all kernel modules
17 #include <linux/kernel.h> // included for KERN_INFO
18 #include <linux/init.h> // included for __init and __exit macros
```

```

19 | 20 | static int __init hello_init(void)
21 | {
22 | printk(KERN_INFO "Hello world!\n");
23 | return 0; // Non-zero return means that the module couldn't be loaded.
24 | }
25 |
26 | static void __exit hello_exit(void)
27 | {
28 | printk(KERN_INFO "Exit module.\n");
29 | }
30 |
31 | module_init(hello_init);
32 | module_exit(hello_exit);
33 |
34 | MODULE_AUTHOR("aggresss <aggresss@163.com>");
35 | MODULE_LICENSE("GPL v2");

```

## Makefile

```

1 | #
2 | # A simple module for helloworld
3 | #
4 | # Copyright (C) 2017 aggresss (aggresss.163.com)
5 | #
6 | # Licensed under GPLv2 or later.
7 | #
8 |
9 | ifneq ($(KERNELRELEASE),)
10 |
11 | MODULE_NAME := test_module
12 | $(MODULE_NAME)-objs := hello.o
13 | obj-m += $(MODULE_NAME).o
14 |
15 | else
16 |
17 | KVER := $(shell uname -r)
18 | KDIR := /lib/modules/$(KVER)/build
19 |
20 | all:
21 |     $(MAKE) -C $(KDIR) M=$(CURDIR) modules
22 | clean:
23 |     @rm -rf .tmp_* *.o.cmd *.o *.ko *.mod.c *.order *.symvers
24 |

```

上面的文件可以在 <https://github.com/aggresss/LKDemo> 的hello\_module 目录中下载。

关于内核模块的编译如果是在本平台编译还是比较简单的，直接在内核模块对应的源文件目录下输入make 就可以编译了，但这样编译出来的模块是在本平台下运行的，因为它使用了当前操作系统的/lib/modules/\$(uname -r)/build 目录来生成当前的模块，并且编译器也是同平台下的。如果想编译在模拟开发板上运行的模块，就需要采用另一种方法，利用Linux 源文件根目录下Makefile 的 SUBDIRS 参数，下面来说明一下操作步骤：

1. 进入到已经编译生成Linux源文件根目录下 例如： linux-4.1.38，前提是已经利用这个目录编译生成过内核目标文件；

2. 执行 export ARCH=arm

export CROSS\_COMPILE=arm-linux-gnueabi-

3. make modules SUBDIRS= xxxx/hello\_module （xxxx为需要编译的模块所在目录）

完成上面的操作后便可以将生成的 \*.ko 文件导入模拟开发板中运行，通过 insmod 命令装载模块，lsmod 命令查看模块，rmmod 命令卸载模块，dmesg 命令查看内核调试信息，下面是演示截图：

```
[root@vexpress nfs]#
[root@vexpress nfs]#
[root@vexpress nfs]#
[root@vexpress nfs]# insmod test_module.ko
[root@vexpress nfs]# lsmod
test_module 530 0 - Live 0x7f00c000 (0)
[root@vexpress nfs]# rmmod test_module
[root@vexpress nfs]# lsmod
[root@vexpress nfs]#
[root@vexpress nfs]#
```

这里需要说明一下：

1.在目标开发板上执行 rmmod 时会提示 **rmmod: can't change directory to '/lib/modules': No such file or directory** 这个错误，是由于没有 /lib/modules/4.1.38 这个目录导致的，在目标开发板上建立一下这个目录即可，是空目录，不需要任何文件。

2.Makefile中的模块名称和依赖文件不要重名，也就是MODULE\_NAME 不能 (MODULE\_NAME)-objs 依赖的源文件的名字重复，这样容易造成循环依赖，导致make会出现如下错误信息：

**make[2]: Circular /root/develop/kernel\_module/helloworld/hello.o <- /root/develop/kernel\_module/helloworld/hello.o dependency dropped.**

更好的方法就是给模块命名的末尾加上 \_module 后缀。

更多内核模块编译相关的知识可以参考：

<http://www.ibm.com/developerworks/cn/linux/l-cn-kernelmodules/>