

前言

为什么需要ARM模拟系统

应用程序的开发

系统开发(BSP)

Qemu是什么?

Qemu的两种模式

Qemu 能做什么? 或者说适合做什么?

在 Ubuntu16.04 系统中, 利用 Qemu 搭建一个ARM虚拟机

使用Qemu虚拟机的几种选择

测试平台

安装交叉编译器

编译内核kernel

制作根文件系统

利用Qemu启动ARM虚拟机

测试HelloWorld应用程序

总结

软件下载地址

前言

前一段时间因为工作需要, 我对ARM模拟器进行了一番调研。调研目的是: 由于项目参与人员比较多, 如果人手一块ARM开发板, 资源比较紧张, 希望能够用**模拟器**来代替。

在调研期间, 花费了很多时间去查资料、测试验证。在实际验证的时候, 发现一个现象: 很多文章虽然给出了或简单、或详细的操作步骤, 但是**大部分都没有写清楚操作的背景、各个软件的版本**, 这就导致**需要把整个文章看完了、把所有的操作步骤都做了一遍, 才明白作者想表达的是什么意思, 操作的目的是什么。**

我觉得, 任何一篇文章, 首先要让读者知道**为什么要读这篇文章**, 或者说读了这篇文章能够**有什么收获**。

如果是操作性比较强的文章, 那么就有必要交代清楚工作平台的**背景**是什么, 要达到的**目的**是什么, **总体步骤**是怎么样的。只有这样, 阅读文章的人在心中首先建立一个宏观的框架, 在理解框架的基础上, 再去实际操作, 这样的话就更容易理解。

当然了, 每个人的学习和阅读习惯都不一样, 上面只是我个人的感受, 或者说我喜欢这样比较有条理的文章, 这样才不至于迷茫。

回到Qemu的主题上来, 这篇文章主要是把调研的结果进行梳理、汇总, 包括如下内容:

公众号【IOT物联网小镇】

为什么需要ARM模拟系统？

Qemu是什么？

Qemu 能做什么？或者说适合做什么？

在 Ubuntu16.04 系统中，利用 Qemu 搭建一个ARM虚拟机操作步骤是什么？

编写一个HelloWorld程序，放到虚拟机中执行。

为什么需要ARM模拟系统

ARM平台的软件开发工作，可以划分为2类：

应用程序的开发

系统开发(内核、文件系统、驱动程序)

应用程序的开发

我们在开发嵌入式项目的时候，一般都是先在x86平台上把大部分的功能开发完成，然后再交叉编译，得到在ARM平台的可执行程序或者库文件。再通过**scp指令**或者**NFS远程挂载**的方式，把这些文件复制到ARM板子上之后执行。

一般而言，应用程序就是利用硬件产品的各种资源、外设，来完成特定的功能，比如：数据采集、控制外部设备、网络传输等等。主要的特征就是与外部的各种设备进行交互。

系统开发(BSP)

系统开发的最终目的是：为应用程序的执行准备一个基本的执行环境，内容包括：**系统引导程序bootloader**，**内核kernel**，**文件系统rootfs**，**系统中所有设备的驱动程序**。在实际的项目开发中，系统开发难度更大一些，一旦开发完成，对于一块板子来说基本上不会轻易变动，代码的使用生命周期更长。

以上这两种分类，主要是从开发工作的内容角度来进行划分的。可以看出：

应用程序开发：灵活性更大、需求变动会更多(产品经理或项目经理经常给你改需求)。

系统软件开发：需求更稳定、很多代码都是官方提供或者开源的，工作内容就是进行定制、裁剪。

对于系统软件开发来说，**如果每次编译出一个bootloader、或者kernel，都上一个ARM开发板进行验证，的确比较麻烦。如果能有一个ARM模拟系统，直接在x86上进行模拟，工作效率就会提高很多。**

Qemu是什么？

Qemu是一个开源的**托管虚拟机**，**通过纯软件来实现虚拟化模拟器**，几乎可以模拟任何硬件设备。比如：Qemu可以模拟出一个ARM系统中的：CPU、内存、IO设备等，然后在这个模拟层之上，可以跑一台ARM虚拟机，这个ARM虚拟机认为自己在和硬件进行打交道，但实际上这些硬件都是**Qemu模拟出来的**。

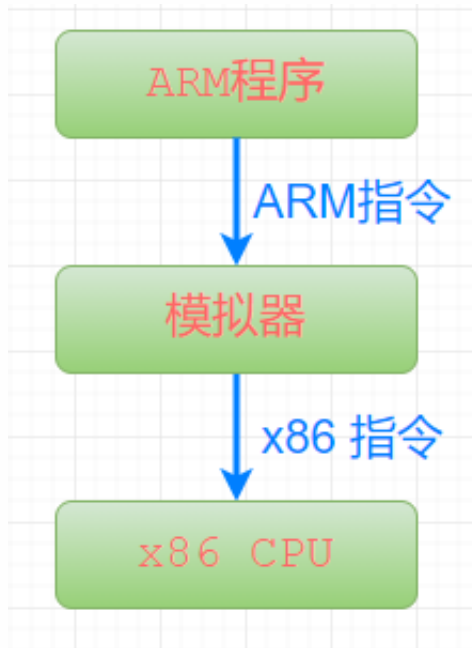


正因为Qemu是纯软件实现的，所有的指令都要经过它的转换，所以性能非常低。所以在生产环境中，大多数的做法都是配合KVM来完成虚拟化工作，因为KVM是硬件辅助的虚拟化技术，主要负责比较繁琐的CPU和内存虚拟化，而Qemu则负责I/O虚拟化，两者合作各自发挥自身的优势，相得益彰。这部分不是重点，就不具体深入介绍了。

Qemu的两种模式

Qemu有**两种执行模式**：

1. 用户模式(User mode): 利用动态代码翻译机制来执行不同主机架构的代码，例如：在x86平台上模拟执行ARM代码，也就是说：我们写一条ARM指令，传入整个模拟器中，模拟器会把整个指令翻译成x86平台的指令，然后在x86的CPU中执行。



2. 系统模式(System mode): 模拟整个电脑系统, 利用其它VMM(Xen, KVM)来使用硬件提供的虚拟化支持, 创建接近于主机性能的全功能虚拟机。



Qemu 能做什么? 或者说适合做什么?

因为Qemu是使用纯软件模拟的, 它的**强项**是模拟那些不涉及到外部的具体硬件设备的场景, 比如:

公众号【IOT物联网小镇】

想学习如何定制bootloader;
想在Arm系统中进行文件系统的裁剪,学习文件系统的挂载过程;
想体验一下如何配置、裁剪linux kernel;
想学习Linux系统中的设备树;
...

以上这些场景中,都非常适合使用Qemu来模拟ARM系统。

在 Ubuntu16.04 系统中,利用 Qemu 搭建一个ARM虚拟机

使用Qemu虚拟机的几种选择

利用Qemu来运行ARM虚拟机,你有2个选择:

1. 简单方式: 直接下载别人编译好的映像文件(包含了内核,根文件系统),直接执行即可。

缺点是: 别人编译好的也许不适合你的需求,没法定制。

2. 复杂方式: 自己下载内核代码、根文件系统代码(例如: busybox),然后进行编译。

优点是: 可以按照自己的实际需求,对内核、根文件系统机型裁剪。

在第2种复杂模式中,又可以有2个选择:

- 2-1. 内核代码、根文件系统代码全部自己手动编译,最后把这些编译结果手动组织在一个文件夹中,形成自己的根目录;
- 2-2. 利用 buildroot 整个框架,只需要手动进行配置(比如: 交叉编译器在本机上的位置、输出路径、系统的裁剪),然后就可以一键编译出一个完整的系统,可以直接烧写到机器!

以上这几种操作方式的选择,可以根据自己的实际需要来选择。如果对构建系统的整个流程已经非常熟悉了,就利用buildroot工具;如果是想更彻底的学习制作一个系统,那就手动一步一步的实际编译、操作一遍,多练几次,你就变成大牛了。

下面,我们就按照2-2的方式,进行实际操作一遍。所有的指令部分,我都直接贴代码,不用截图,这样方便复制。

测试平台

我的工作电脑是Win10,通过VirtualBox安装了Ubuntu16.04虚拟机,64位系统。

下面的操作在Ubuntu16.04虚拟机中可以顺利编译,当然,一些基本的工具(例如: build-essential, make等基础工具软件这里就不详述了)。

安装交叉编译器

交叉编译器的作用就不需要详细解释了,因为我们是在x86平台上进行编译,而运行的平台是ARM系统,这2个平台的指令集不一样,所以需要交叉编译得到ARM系统上可以执行的程序。

公众号【IOT物联网小镇】

```
sudo apt-get install gcc-arm-linux-gnueabi
```

验证安装结果

```
dpkg -l gcc-arm-linux-gnueabi
```

显示如下：

ii	gcc-arm-linux-gnueabi	4:5.3.1-1ubuntu1	amd64	GNU C compiler for the armel architecture
----	-----------------------	------------------	-------	---

有些文章建议自己下载交叉编译器，然后手动设置环境变量。我实际操作了一下，手动下载的交叉编译工具链在编译内核的时候报错，所以还是[建议直接用apt-get直接安装](#)。

编译内核kernel

内核kernel的作用也是不言而喻的，就相当于我们的Windows操作系统，没有这个操作系统，硬件就是一堆废铁。当系统启动的时候，会把内核加载到内存中，然后从内核的入口地址开始执行。

1. 下载内核

版本：linux-4.14.212.tar。

在文末，我会列出所有的软件包下载地址。

2. 使用现成的vexpress开发板子的config文件

```
make CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm vexpress_defconfig
```

这个操作，会把vexpress_defconfig作为配置文件保存为.config，下面在编译内核时就根据这个config中的配置进行编译。

如果需要对内核进行裁剪，执行：

```
make menuconfig
```

根据自己的实际需要，对内核进行定制。比如：可以配置网络和NFS，在系统启动的时候就自动挂载宿主机中的某个目录。

3. 编译内核

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

编译得到内核文件arch/arm/boot/zImage，Qemu启动时需要指定使用这个映像文件。

制作根文件系统

内核在启动之后、执行到最后步骤时，需要[挂载根文件系统](#)，然后执行文件系统中指定的执行程序，例如：/etc/rc.local。

如果没有跟文件系统，那么内核在执行到最后就提示：panic...。

1. 下载busybox

公众号【IOT物联网小镇】

版本: busybox-1.20.2.tar.bz2。

2. 创建rootfs根目录

```
mkdir -p rootfs/{dev,etc/init.d,lib}
```

3. 把busybox-1.20.2中的文件复制到rootfs根目录下，主要是一些基本的命令

```
cp busybox-1.20.2/_install/* -r rootfs/
```

4. 把交叉编译工具链中的库文件复制到rootfs根目录的lib文件夹下

```
sudo cp -P /usr/arm-linux-gnueabi/lib/* rootfs/lib/
```

5. 制作根文件系统镜像

根文件系统镜像就相当于一个硬盘，就是把上面rootfs根目录中的所有文件复制到这个硬盘中。

(1) 生成512M大小的磁盘镜像

```
qemu-img create -f raw disk.img 512M
```

(2) 把磁盘镜像格式化成ext4文件系统

```
mkfs -t ext4 ./disk.img
```

(3) 将rootfs根目录中的所有文件复制到磁盘镜像中 操作步骤是：创建挂载点-挂载-复制文件-卸载。

```
mkdir tmpfs  
sudo mount -o loop ./disk.img tmpfs/  
sudo cp -r rootfs/* tmpfs/  
sudo umount tmpfs
```

(4) 使用file指令检查一下

```
file disk.img
```

```
disk.img: Linux rev 1.0 ext4 filesystem data, UUID=f3b914f1-f5cd-4494-962b-97fc688afe9f (extents) (large files) (huge files)
```

利用Qemu启动ARM虚拟机

1. 启动虚拟机

这个命令有点长，测试时建议直接复制、粘贴。

公众号【IOT物联网小镇】

```
qemu-system-arm -M vexpress-a9 -m 512M -kernel ./linux-4.14.212/arch/arm/boot/zImage -dtb ./linux-4.14.212/arch/arm/boot/dts/vexpress-v2p-ca9.dtb -nographic -append "root=/dev/mmcblk0 rw console=ttyAMA0" -sd disk.img
```

2. 停止虚拟机

在Ubuntu另一个终端窗口中，通过`killall`指令来停止。

```
killall qemu-system-arm
```

当然，也可以用`ps`指令找到`qemu-system-arm`的进程号，然后通过`kill -9`来停止虚拟机。

测试HelloWorld应用程序

1. 在Ubuntu任意一个目录，编写HelloWorld可执行程序`hello.c`:

```
#include <stdio.h>
int main()
{
    printf("HelloWorld! \n");
    return 0;
}
```

2. 交叉编译`hello.c`，得到可执行程序`hello`:

```
arm-linux-gnueabi-gcc hello.c -o hello
```

通过`file`指令，查看一下`hello`程序:

```
file hello
```

```
hello: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.3, for GNU/Linux 3.2.0, BuildID[sha1]=765500b8549a507268deede47639aa98d09d80d6, not stripped
```

3. 通过`kill`命令停止虚拟机。
4. 把`hello`可执行程序复制到磁盘镜像`disk.img`中
操作步骤是: 挂载-复制文件-卸载。

```
sudo mount -o loop ./disk.img tmpfs/
cp hello tmpfs/
sudo umount tmpfs
```

5. 执行`hello`程序

再次启动虚拟机，此时可以在根目录下面看到`hello`文件，直接执行即可看到输出结果。

公众号【IOT物联网小镇】

总结

在以上的操作步骤中，我们把一个ARM系统在启动应用程序之前，所需要的程序都手动编译、操作了一遍。看一遍很容易就明白，亲手操作一遍印象会更深刻。

这里的操作过程有些还需要继续深入，比如：在系统启动之后，自动挂载宿主机(Ubuntu系统)中的某个文件夹，这样就可以把hello等可执行程序复制到挂载目录中，然后在ARM系统中直接执行了，而不用再执行下面在一连串的操作(停止虚拟机-挂载磁盘镜像-复制文件-卸载-启动虚拟机)。

最后，希望这篇总结能给你带来小小的收获和提升！

软件下载地址

1. linux-4.14.212.tar.xz

链接：<https://pan.baidu.com/s/1d8RxjMkYQhPtZgiybD8Gw>

提取码：b6ft

2. busybox-1.20.2.tar.bz2

链接：https://pan.baidu.com/s/1oPeH7juEWuFR6y1Qpna_BA

提取码：9kh6

【原创声明】

作者：道哥(公众号: IOT物联网小镇)

知乎：道哥

B站：道哥分享

掘金：道哥分享

CSDN：道哥分享

如果觉得文章不错，请转发、分享给您的朋友。

我会把十多年嵌入式开发中的项目实战经验进行总结、分享，相信不会让你失望的！

长按下图二维码关注，每篇文章都有干货。



公众号【IOT物联网小镇】

转载：欢迎转载，但未经作者同意，必须保留此段声明，必须在文章中给出原文连接。

推荐阅读

- [1] [原来gdb的底层调试原理这么简单](#)
- [2] [生产者和消费者模式中的双缓冲技术](#)
- [3] [深入LUA脚本语言，让你彻底明白调试原理](#)
- [4] [一步步分析-如何用C实现面向对象编程](#)