龙芯 QEMU 模拟器使用介绍

目录

一、 什么是 QEMU

1.1 QEMU 模拟器简介

QEMU 是一个通过软件来模拟机器架构的模拟器。在这样的一个模拟器中，一个软件变量可能就用来模拟一个硬件上的寄存器，而硬件上的各种运算也会通过软件运算来模拟。本课程提供给同学们的 QEMULoongson 就是用来模拟龙芯开发板的一个模拟器，功能与龙芯基本相同，包括 CPU、接口、网络、内存地址空间等。同学们可以将自己写的操作系统运行在 QEMU 上，方便调试自己的代码。

使用模拟器进行调试的优点是：QEMULoongson 可以接入 GDB，通过 GDB 来单步调试、设置断点、输出系统寄存器内容、输出地址空间内容等等，方便大家定位 bug 和判断 bug 原因。

1.2 学习使用模拟器

学习使用 QEMU 和 GDB 会花一点时间，但是会大大降低同学们解决代码 bug 的难度。由于 QEMU 上的功能与输出基本和开发板相同，所以同学们可以在 QEMU 上调试代码，通过之后再上开发板调试。这样不仅可以减少反复 sd 卡拷贝的过程，也可以利用模拟器与 gdb 的连接，更方便的调试代码中的 bug。

学习使用模拟器对于同学们未来的研究生涯也会有帮助。由于硬件平台无法修改，而且调试复杂，所以使用软件模拟器来进行硬件和结构方面的研究会是一种常用的手段。在本课程中了解模拟器的概念和用法，对同学们未来的研究也会有一定帮助。同时在模拟器与 gdb 的配合使用中，同学们也可以熟悉 Linux 的代码调试利器：gdb 的使用。

但是必须要提示同学们的是，模拟器只能帮助代码的调试，而各个 project 的设计必须由同学们自己仔细思考，一个良好的代码设计才是又快又好完成代码的最重要基础。

二、 需要的工具

在课程网站上我们提供的 QEMULoongson.rar 中，包含了 QEMU 的运行程序（bin 文件夹下的程序）和运行脚本（run_pmon.sh），以及和开发板上相同功能的 PMON（bios 文件夹下的 gzram.bin）。此外同学们可以自己安装 gdb-multiarch，由于龙芯平台使用 MIPS 架构，所以一般的 gdb 并不能支持，需要使用支持多架构的 gdb-multiarch。如果同学们在 PO 中使用的是我们提供的带交叉编译环境的 ubuntu 镜像,那么其中已经安装好了 gdb-multiarch。如果同学的环境下没有安装,可以通过输入 `apt-get install gdb-multiarch` 命令安装。如果同学们使用的 Linux 是 CentOS 或 Fedora 等，可以使用 QEMULoongson.rar 压缩包中包含的 mipsel-linux-gdb 来实现相同功能。

三、 使用方法

在这一章中，我们将介绍 QEMU 模拟器的使用方法。

3.1 制作 USB 镜像盘

类似于开发板，我们需要一个 SD 卡来存储同学们自己编写的操作系统代码，并由 PMON 读进内存。但是由于 QEMULoongson 模拟器的功能限制，我们这里使用一个虚拟的 USB 盘来代替 SD 卡。

首先，我们需要制作一个大一点的空磁盘镜像，输入指令

```
dd if=/dev/zero of=disk bs=512 count=1M
```

这里之所以要先制作一个 512MB 容量的空盘，是为了防止同学们直接用代码来制作磁盘镜像，使得磁盘镜像过小，一旦读取超过磁盘大小的偏移位置就会报错。

接着我们将同学们的代码写入磁盘，类似于 P1 中介绍的 make floppy 操作，即输入以下命令：

```
./createimage -extended bootblock kernel
```

```
dd if=image of=disk conv=notrunc
```

注意第二条命令的 if 是 createimage 出来的 image 文件，of 后面是上一步中建好的空磁盘镜像，请同学们自己使用的时候注意文件路径。

```
stu@stu-VirtualBox:~/QEMULoongson$ dd if=image of=disk conv=notrunc
2+0 records in
2+0 records out
1024 bytes (1.0 kB) copied, 0.000607305 s, 1.7 MB/s
stu@stu-VirtualBox:~/QEMULoongson$
```

输出入上图所示。至此，我们就成功的把自己编写的操作系统代码写入了一个供 QEMU 使用的 USB 盘镜像。

3.2 启动 QEMU 模拟器

接下来，使用我们提供的 run_pmon.sh 脚本，同学们就可以启动 QEMU 模拟器并搭载上上一小节中建好的 USB 盘镜像了。

run_pmon.sh 中的内容如下（前面的一些开头带有#的脚本不会生效，只是备用）：

```
SERIAL=2 ./qemu/bin/qemu-system-mipsel -M ls1c -vnc 127.0.0.1:0 -kernel ./bios/gzram -gdb
tcp::50010 -m 32 -usb -drive file=disk,id=a,if=none -device usb-storage,bus=usb-bus.1,driv
e=a  -serial stdio  "$@"
```

其中-kernel 后面的文件是我们提供的 pmon 文件，-drive file=后面的文件是 USB 镜像，请同学们注意这两个参数的文件路径是否正确。

检查无误后，就可以启动 run_pmon.sh 脚本了，但是在此之前，同学们需要注意 QEMU 执行程序是否有可执行权限。需要输入以下命令：

```
chmod +x qemu/bin/qemu-system-mipsel
```

然后，我们就可以通过以下命令启动运行脚本了。

```
sh run_pmon.sh
```

经过一段前面的输出之后，同学们应该就看到和开发板上相同的 PMON 命令提示符了，如下：

```
CPU Loongson 1C300A QEMULoongson V2.0 @ 240.00 MHz / Bus @ 120.00 MHz
Memory size  32 MB ( 32 MB Low memory,   0 MB High memory) .
Primary Instruction cache size 16kb (32 line, 4 way)
Primary Data cache size 16kb (32 line, 4 way)

BEV in SR set to zero.
PMON>
```

至此，QEMU 模拟器的启动已经完成，同学们可以在熟悉的 PMON 命令符下输入 loadboot 命令，看看和在开发板上是否完全一样:)

3.3 连接 gdb 工具

通过前面的步骤，我们已经可以看出，QEMU 模拟器的功能和开发板上基本一样。但是只做到这一点并不能让同学们更好的调试自己的代码，所以连接 gdb 是必要的。

在 QEMU 已经启动的情况下，在另一个窗口中输入 gdb-multiarch 命令，看到以下输出即进入 gdb 程序：

```
stu@stu-VirtualBox:~$ gdb-multiarch
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>.
(gdb)
```

然后在 gdb 命令提示符后，我们就可以使用一些 gdb 的命令了。首先我们先来连接正在运行的 QEMU 模拟器。输入下面的命令分别设置架构名称和远程端口：

```
set arch mips
```

```
target remote localhost:50010
```

可以看到下面的输出：

```
(gdb) set arch mips
The target architecture is assumed to be mips
(gdb) target remote localhost:50010
Remote debugging using localhost:50010
0x8004503c in ?? ()
(gdb)
```

这样就已经成功的连接上 QEMU 模拟器了。要注意的是，使用 target remote 连接成功之后，QEMU 模拟器会进入暂停执行的状态，我们可以在 gdb 命令提示符后面输入 c 命令，QEMU 模拟器就可以继续运行，输出下图所示：

```
(gdb) c
Continuing.
```

后面可能会继续打印一些 warning，但是同学们切回 QEMU 的界面，应该可以发现 QEMU 模拟器已经可以继续运行，并接收输入了。

3.4 gdb 调试常用命令

通过上一步的操作，同学们已经将 gdb 和 QEMU 模拟器连接了起来，可以使用 gdb 来对 QEMU 模拟器进行调试了。这里介绍一些调试中常用的命令和它们的效果，同学们也可以继续钻研和讨论更多的用法。

（1） 设置断点命令：b。范例如下：

```
(gdb) b *0xa0800000
warning: GDB can't find the start of the function at 0x8004503c.
Breakpoint 1 at 0xa0800000
(gdb) c
Continuing.

Breakpoint 1, 0xa0800000 in ?? ()
(gdb)
```

注意 b 命令后面参数中的地址，前面要带一个*号。

设置断点成功之后就会显示 Breakpoint 1 at XXXX，这里的 1 是断点的编号，再设置一个断点就会变成 2 号。

使用 c 继续执行之后，QEMU 模拟器就会在运行到断点位置是停下，停下之后，同学们就可以使用后面介绍的一些指令来查看当前的状态了。

（2） 单步运行：si。范例如下：

```
(gdb) si
warning: GDB can't find the start of the function at 0xa0800000.
warning: GDB can't find the start of the function at 0xa0800004.
0xa0800004 in ?? ()
(gdb)
warning: GDB can't find the start of the function at 0xa0800008.
0xa0800008 in ?? ()
(gdb)
warning: GDB can't find the start of the function at 0xa080000c.
0xa080000c in ?? ()
(gdb)
warning: GDB can't find the start of the function at 0x8007b980.
0x8007b980 in ?? ()
(gdb)
```

注意：范例中后面的几个 gdb 命令提示符后面没有输入 si，是因为我在这里使用敲了回车。在 gdb 中，直接敲回车键相当于重复之前的命令。

在范例中我们可以看到，代码从 0xa080000c 之后跳到了 0x8007b980，通过单步执行我们有效的跟踪了代码执行的跳转过程。

（3） 查看寄存器内容：i r。范例如下：

```
(gdb) i r
          zero       at       v0       v1       a0       a1       a2       a3
 R0   00000000 fffffc0 8000bb38 a0800000 a0800080 801ffd28 8000b3d8 8000b3cc
           t0       t1       t2       t3       t4       t5       t6       t7
 R8   8007b980 00000001 00000000 800c0000 800c0000 800b3eb4 8008b694 ffffffff
           s0       s1       s2       s3       s4       s5       s6       s7
 R16  00000200 0000000f 00000200 8000b938 80090f18 8009a7d4 8000bb68 00000000
           t8       t9       k0       k1       gp       sp       s8       ra
 R24  00000170 8000b430 00000000 00000000 80850000 8000b928 00000001 a0800014
           sr       lo       hi      bad    cause       pc
      00000000 00000000 00000000 00000000 40008000 8007b980
          fsr      fir
      00000000 00739300
(gdb)
```

通过这样的命令，我们可以输出所有寄存器的内容。

也可以在 i r 后面空格再输入寄存器名，则只输出指定寄存器的内容。

（4） 查看内存内容：x，命令格式为 x/nfu [addr]（n 是内存单元个数，f 是显示格式，u 是内存单元大小），范例如下：

```
(gdb) x/10i 0xa0800000
   0xa0800000:  lui     t0,0xa080
   0xa0800004:  lw      t0,156(t0)
   0xa0800008:  lui     a0,0xa080
   0xa080000c:  jalr    t0
   0xa0800010:  addiu   a0,a0,128
   0xa0800014:  nop
   0xa0800018:  lui     t0,0xa080
   0xa080001c:  lw      t0,152(t0)
   0xa0800020:  lui     a0,0xa080
   0xa0800024:  lw      a0,164(a0)
(gdb) x/20x 0xa0800000
0xa0800000:     0x3c08a080      0x8d08009c      0x3c04a080      0x0100f809
0xa0800010:     0x24840080      0x00000000      0x3c08a080      0x8d080098
0xa0800020:     0x3c04a080      0x8c8400a4      0x24050200      0x3c060001
0xa0800030:     0x0100f809      0x34c64000      0x00000000      0x3c08a080
0xa0800040:     0x8d0800a8      0x0100f809      0x00000000      0x03e00008
(gdb)
```

通过改变 x 命令后面的参数，可以灵活的查看内存中的内容，比如范例中的 10 和 20 就是查看的指定地址之后的内存长度，i 和 x 则代表将内存内容以汇编指令的格式或者十六进制数据的格式打印出来。

（5） 退出 gdb：q

四、 总结

通过学习使用 QEMULoongson 模拟器，同学们可以将自己编写的操作系统代码在模拟器上运行。并且通过连接 gdb，更方便的进行调试。但是请同学们注意的是，我们的作业提交依然以在开发板上运行的结果为准，请同学们在 QEMU 上调试通过后一定在开发板上也运行一下，确认没有问题再提交代码。

而在学习 QEMU 模拟器的过程中，同学们可以在熟悉模拟器使用的同时，学习 gdb 的使用，这不仅能协助本课程各个 projects 的完成，我们相信它更对同学们未来的学习和研究生涯有重要帮助，请同学们善加利用。同学们也可以挖掘出模拟器和 gdb 更好的运用手段，反馈给老师们和同学们，帮助大家共同进步。