(挑战性课程用)简明QEMU手册

目的

这份手册, 就如何在mipssim下添加pflash支持和如何使用qemu-thumips进行简要的说明, 主要针对参加挑战性课程的同学. 修改的过程本身比较简单, 但如果自己探索的话还是需要花不少时间, 也可能会踩到很多坑(大神请无视此句), 另一方面, qemu-thumips没有给出太清晰的文档, 我希望进一步详细说明, 这也是我写这份文档的动机.

试验环境

ubuntu 14.04

关于qemu的获取与安装

官方qemu

由于需要对qemu进行修改, 因此我们需要获取qemu的源代码. 在下载或clone代码时, 请务必注意branch的选取, **最好选择stable-2.0分支**, 这与目前ubuntu源中qemu的版本相同. 经实际测试, mips-ucore在qemu 2.6中无法正确运行, 表现为进入用户态并进行三次键盘输入之后卡死, 因此强烈推荐选择stable-2.0分支.

qemu-thumips

qemu-thumips是chyh学长修改过的qemu, 加入了一些新功能, 主要是mipssim下的指令集裁剪和只读flash的模拟. 从chyh1990/qemu-thumips 获取即可.

关于安装

细碎的经验

安装遵循经典的

```
./configure
make
make install
```

因此不再赘述.

要说明的是在进行`./configure`的时候, 可以使用`--target-list=mipsel-softmmu`参数选择只编译mipsel架构的qemu, 节省编译时间.

另外, `--prefix=[absolute path]`参数可以指定qemu的安装路径.

在`./configure`之前输入`./configure --help`查看帮助总是比较好的习惯

关于qemu-thumips

在编译qemu-thumips的过程中可能出现种种编译错误, 但这些都是可以自行解决的.

只有一个链接错误需要特别强调, 直接使用现有的Makefile有可能会出现一个动态链接库的链接错误, 这时, 只需要在`Makefile.target`的第37行, 将:

```
ifndef CONFIG_HAIKU
LIBS+=-lm
```

```
endif
```

改为

```
ifndef CONFIG_HAIKU
LIBS+=-lm -lrt
endif
```

即可

pflash的添加

这里的pflash指的就是thinpad上的NOR flash, 但在mipssim中并没有添加对pflash的支持, 可以仿照其他架构下的实现方式在mipssim中进行添加.

以及由于qemu支持多个架构, 所以请确定你正在修改mips架构下的文件, 而不是其他架构下的同名文件, 这句话适用于全文

找到`[qemu code]/hw/mips/mips_mipssim.c`, 在文件开头添加:

```language=C
# include "hw/block/flash.h"
```

这其中包含了对于`pflash`的声明.

接下来, 在该文件当中找到`mipssim`的初始化函数`` `mips_mipssim_init` ``, 在函数的最后,

```
DriveInfo *dinfo = drive_get(IF_PFLASH, 0, 0); if (!(pflash_cfi01_register(0x1E000000, NULL, "flash.img", 0x08000000, dinfo->bdrv, (128 << 10), 64, 2, 0, 0, 0, 0)))
```

向模拟器注册`pflash`. 其中`` `pflash_cfi01_register` ``是`pflash`的注册函数, 它的声明是

```
pflash_t *pflash_cfi01_register(hwaddr base, DeviceState* qdev, const char *name, hwaddr size, BlockDriverState* bs, uint32_t sector_len, int nb_blocs, int width, uint16_t id0, uint16_t id1, uint16_t id2, uint16_t id3, int be);
```

其中需要关注的有:

* `` `base` ``是`pflash`的物理起始地址

在我们当前使用的`ucore`(就是说你的`ucore`也许不一样)当中, `flash`的虚拟地址约定为`0XBE`

* `` `sector_len` ``是块大小.

* `` `nb_blocs` ``是块的数量.

* `width`是一次读写的字节数.

* `bs`可以用于指定外部的文件作为flash，具体的操作请见[后文](#pflash_usage).

(稍加注意的话，就会发现其他模拟设备也是在这个函数当中进行注册的，例如串口)

pflash的使用

有两种使用pflash的策略，如何选用取决于是否需要对pflash中的数据进行永久保存.

不进行永久保存

选用此种策略，那么对于注册pflash时的`bs`参数，取NULL即可. 启动qemu的命令也与原来

```
qemu-system-mipsel -M mipssim -m 32M -serial stdio -kernel obj/ucore-kernel-initrd
```

进行永久保存

选用此种策略，qemu的代码改动请参考上文.

然后，生成用于模拟pflash的文件：

```
dd if=/dev/zero of=flash.raw bs=8M count=1
```

修饰一下这个文件的大小：

```
truncate -s 8M flash.raw
```

正式进行模拟：

```
qemu-system-mipsel -M mipssim -m 32M -serial stdio -kernel obj/ucore-kernel-initrd -drive if=pflash,id=pflash,format=raw,file=flash.raw
```

这样，ucore对flash的操作就会被记录在文件`flash.raw`当中

pflash的实现位置

这并不是手册的重点，但仍然指出是在文件`pflash_cfi01.c`当中，当中主要实现了pflas

使用qemu-thumips进行模拟

这部分主要针对在计算机组成原理课上进行挑战性课程项目的同学，由于这门课挑战性项目的最

使用指令裁剪功能

指令裁剪是一个可配置的功能，只需要在qemu的__运行路径__下创建一个名为`thumips_ins

ADDIU 001001ssssstttttiiiiiiiiiiiiiiii ADDU 000000sssssttttddddd00000100001

需要说明的是，只有'0'和'1'的字符会被检查，这意味着指令中的's', 't', 'i', 'd'只

指令裁剪实现

这部分只是我阅读代码的结果，因此可能会有错误.

分析的思路很简单，就是从打开`thumips_insn.txt`的代码入手，逐步追踪.

指令裁剪相关功能的代码都实现在`thumips.h`和`thumips.c`当中，其中`load_thumips

除此之外，`thumips.c`中还实现了`verify_inst`，`check_thumips`，`print_opc

指令翻译的具体的过程，在`translate.c`当中的`decode_opc`函数当中实现，要在这里进

int i = check_thumips(ctx->opcode); / *printf("i=%d\n", i);* / if(i < 0){
PRINT_ERR_INSTR("unknown"); MIPS_INVAL("major opcode"); generate_exception(ctx,
EXCP_RI); return; } ```

当然稍作改动, 这也能够成为一个通过黑名单控制合法指令的功能, 只需将判断中的`i < 0`改为`i >= 0`即可.

致谢

在对qemu的探索当中, 我必须感谢张宇翔同学, 如果没有他的帮助, 也许我现在仍然在和莫须有的bug作斗争.

"愿大腿与你同在. "

——来自小白的善意