QEMU1.3.0 的源码分析一 ：源码目录简介

最近在研究 QEMU，读了一些 QEMU 的源码，因为涉及的东西比较多，找到的资料又都比较破碎，不太完整。所以将最近的成果总结一下。

相比其他的开源软件来说，QEMU 源码下面目录比较多，下面就先把这些目录的内容大致整理一下。
docs/ 包含了一些文档，说实话，对初学者来说，读这些文档压根没有头绪
hw/ 包含了所有支持的硬件设备
include/ 包含了一些头文件
linux-user/ 包含了 linux 下的用户模式的代码
target-XXX/ 包含了 QEMU 目前所支持 guset 端的处理器架构。包 括：
alpha,arm,cris,i386,lm32,m68k,microblaze,mips,openrisc,ppc,s390x,sh4,sparc,unicore32,xtensa. 此处的 XXX 就是指这其中的一种架构。包含的代码的主要功能是将该 guest 架构的指令翻译成 TCG OP 代码。也就是 target-arm 下的代码就是将 arm 架构的指令翻译成 TCG OP。这些目录占了源码目录的很大一部分。
tcg/ 包含了动态翻译工具 tcg 的源码部分，主要是将 TCG OP 转化为 host binary 的部分。这个目录下也包含了多个架构名字命名的目录，每个目录下存放着针对该架构的代码。后续会详细介绍。
test/ 从名字上可以看出，应该是存放测试部分的代码，但是目前这部分代码还没读。

QEMU1.3.0 源码分析之二：TCG

TCG 是 Tiny Code Generator 的简称，它之前是一个后端编译器，现在是作为一个动态翻译器来使用。在 QEMU 中，它主要用来将虚拟出来的系统的指令转化成真正硬件支持的指令中，从中间代码到硬件支持的机器代码的过程。前端的将指令翻译成中间代码的过程，是一个反汇编的过程。

反汇编的过程的源码的主要地址：qemu source code/target-XXX。此处的 XXX 指的是模拟出来的系统的架构。

TCG 的源码的位置是：qemu source code/tcg。这个目录下有很多文件夹，每个文件夹都代表一个目标架构。这里的目标架构指的是真正的硬件架构，也就是说运行 QEMU 的架构。

在 qemu source code/tcg 目录下，有一个 README 文件，介绍了 tcg 的主要内容。

在 qemu source code/tcg/arm 目录下，只有两个文件，实现了生成 arm 架构的内容。tcg-target.c 和 tcg-target.h 两个文件。和 arm 同 级目录的 ia64,hppa,ppc,s390,i386,mips,ppc64,sparc,tci 等目录下也是同样的名字的两个文件，当然文件的内容 并不相同。关于这两个文件的内容，README 是这么描述的：tcg-target.h contains the target specific definitions. tcg-target.c contains the target specific code.

动态翻译只是在必要的时候才进行翻译，而尽可能的将时间花费在执行 host code 上。TB(Translation Block)翻译之后得到的 host code 会存放在 code cache 中，因为有很多 TB 会被重复执行，所以这样会达到更加的效果。

QEMU1.3.0的源码分析三：user model 之 linux

从源码目录来看，user model 有两块内容 bsd-user 和 linux-user。我主要研究了下 linux-user 这种情况。

首先要提一下通常容易关注的焦点，linux-user 下的函数入口点：/源码目录/linux-user/main.c 中的

Line:3388　int main(int argc, char **argv, char **envp).

找到了入口函数，就可以根据这个 main 函数中的调用关系来看看这个情况下的主要执行流程和动作了。

```
int main(int argc, char **argv, char **envp)
{
    module_call_init(MODULE_INIT_QOM);
    qemu_cache_utils_init(envp);

    /*初始化了 tcg 的相关部分，包含了 cpu 动态转化的一些初始化操作。*/
    tcg_exec_init(0);
    cpu_exec_init_all();

    /*包含了虚拟 cpu 的初始化*/
    env = cpu_init(cpu_model);
    /*加载可执行程序，即 Guest code*/
    ret = loader_exec(filename, target_argv, target_environ, regs, info, &bprm);

    target_set_brk(info->brk);
    /*系统调用初始化*/
    syscall_init();
    /*信号初始化*/
    signal_init();

    /*此函数是主要的循环体，通过这个函数来实现对指令的动态翻译，并且执行翻译之后的 Host Code。
    通过最终调用 cpu_gen_code()函数（位于 translate-all.c 文件中）来实现动态翻译，其中调用了两个关键
     函数。一个关键函数是 gen_intermediate_code()函数（位于 target-arm/translate.c，此处以 guest 指
     令集为 arm 为例），这个函数的主要功能是根据 Guest Code 生成 TCG Operations。另外一个重要的函数
     是 tcg_gen_code()函数(位于 tcg/tcg.c)，这个函数主要是把 TCG Operations 转化成 Host code。*/
    cpu_loop(env);
    /* never exits */
    return 0;
}
```

下面来分析下刚才介绍的重要函数 cpu_loop(). cpu_loop()函数在 linux-user/main.c 中有多个版本，区别在于参数，参数是不同的 cpu state，下面举例仍然以 arm 为主。

```
void cpu_loop(CPUARMState *env)
{
    int trapnr;
    unsigned int n, insn;
    target_siginfo_t info;
    uint32_t addr;
```

```
    for(;;) {
        cpu_exec_start(env);
        trapnr = cpu_arm_exec(env);
       cpu_exec_end(env);
       ..............
}
```

可以看到 for 循环里有三个函数调用，分别是 cpu_exec_start，cpu_arm_exec，cpu_exec_end。其中最重要的 cpu_arm_exec 函数，通过 target-arm/cpu.h 中的宏定义#define cpu_exec cpu_arm_exec 调用了 cpu-exec.c 文件中的 cpu_exec()函数。

cpu_exec()是整个 qemu 中的一个重要函数，它负责整个核心的从 guest code 到 host code 的翻译和执行。

cpu_exec()首先会去调用 tb_find_fast()，tb_find_fast()会判断取回来的 tb 是否合法，如果不合法会去调用 tb_find_slow()函数。tb_find_slow()会试图通过物理 mapping 去寻找 tb，如果寻找失败则会调用 tb_gen_code()去翻译代码。

cpu_exec()函数调用 tb_find_fast()之后会调用 tcg_qemu_tb_exec()去执行所找到的 tb。最后再调用 cpu_exec_nocache()去执行剩下的代码。

References：

http://my.oschina.net/shinn/blog/94278
http://my.oschina.net/shinn/blog/94280
http://my.oschina.net/shinn/blog/95059