# QEMU日常填坑

## 前言

问题源于我想在KVM的加持下来加速用QEMU调试Linux的虚拟机。刚开始我还没遇到这个问题，突然有一天我需要在start\_kernel()这个函数下断点。发现断不下来。但是系统启动后，对任意函数下断点又没有问题。问题就变成了在KVM的加持下对start\_kernel()下断点无效。

## 背景知识

### QEMU

QEMU是一套由法布里斯·贝拉(Fabrice Bellard)所编写的以GPL许可证分发源码的模拟处理器，在GNU/Linux平台上使用广泛。Bochs，PearPC等与其类似，但不具备其许多特性，比如高速度及跨平台的特性，通过KQEMU这个闭源的加速器，QEMU能模拟至接近真实电脑的速度。（来源-百度百科）

### QEMU部分代码分析

QEMU在使用KVM方案后，kvm\_set\_user\_memory\_region()函数通过KVM API设置虚拟机的内存和主机的内存映射关系。本次调试的虚拟机内存设置为4G,cpu核数为1。

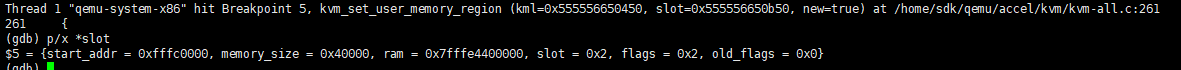


图1

如图1所示我们看到一段虚拟机的内存0xfffc0000是和主机内存0x7fffe4400000做的映射。0xfffc0000位置用于放置bios代码。

下面是我找到一些我们后面要用的虚拟机和主机映射的地址范围。

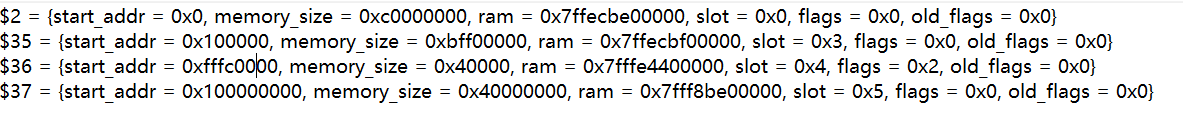


图2

图2中的其中比较有意思的是$37这组数据。上面说过虚拟机只有4G物理内存，为什么会有4G-5G这一段物理内存那？这个问题是PCI BUS总线地址原因。PCI BUS总线地址占到物理内存的3G-4G位置上，我们的那段3G-4G内存地址就映射到了4G-5G的内存地址。在Linux的dmesg中看e820的布局也能看到这个现象。e820中没有3G-4G的物理地址，代替它的是4G-5G的物理地址。

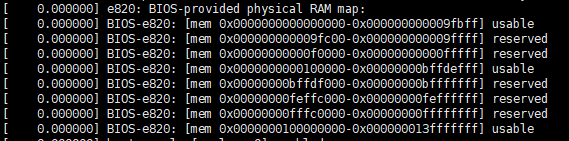


图3

图2的内存布局是为了方便我们下一步验证的工作。QEMU加载Linux内核代码在load\_linux()中实现的。部分代码如下：

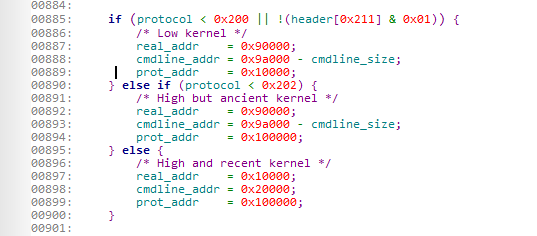


图4

图4中的代码告诉QEMU把一个linux的内核加载到物理内存的位置。调试的内核是Linux-4.13-2版本，物理内存位置在897行。real\_addr的部分是放置Linux内核header.S(arch/x86/boot)的代码。一个压缩的内核例如bzImage由两部分组成setup和kernel.其中kernel是压缩了，setup部分用于启动和解压。图4 real\_addr用于放setup实模式的代码。prot\_addr用于放置setup保护模式的代码。

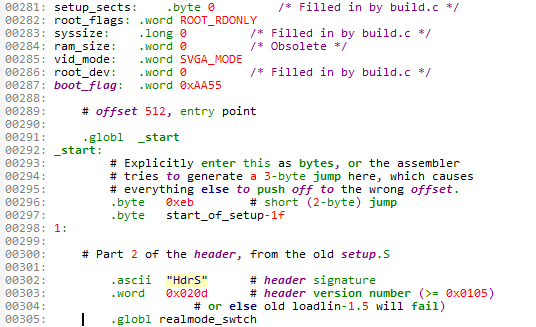


图5

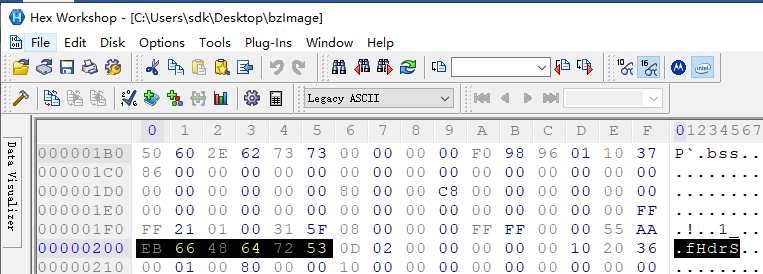


图6

图5是代码，图6是生成后内核。header.S很多功能例如定义了setup的大小等重要信息。header.S中有很多关键的信息用于确定后面压缩的内核在那。

知道了QEMU把Linux加载到哪，下面就是QEMU什么时候加载Linux内核了。通过图2找到虚拟机0x10200物理地址在主机内存的位置，下一个watch断点我们找到如图7所示



图7

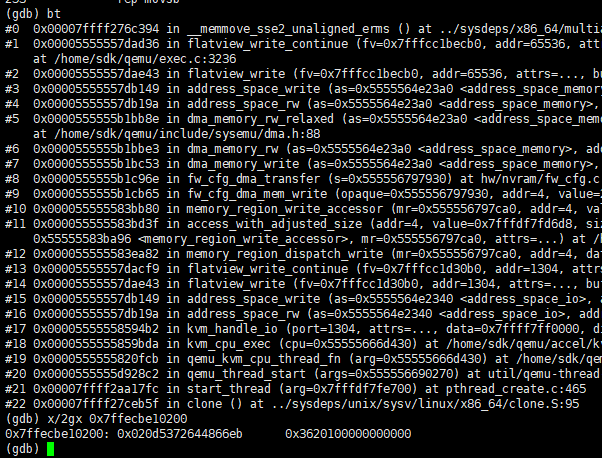


图8

图8已经看到QEMU中运行的bootloader代码把bzImage加载到指定的内存中。下面介绍一种不是完美方法来查看虚拟机中当前CPU运行的状态。首先对kvm\_getput\_regs()中有对虚拟机中CPU寄存器的获取



图9

图9 为下断点获取((CPUX86State \*) 0x5555566756e0)它保存，

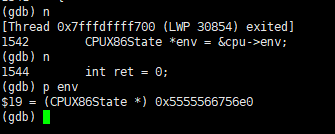


图10

此函数是在KVM中的虚拟机执行前执行。所以获取当前CPU的CPUX86State比较好。

继续回到上面看图8中此时虚拟机中断时候的寄存器和内存状态。

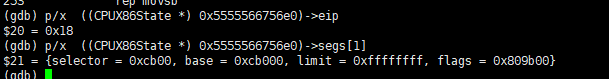


图11

段寄存器是保存在segs数组里面的其中cs的位置如下

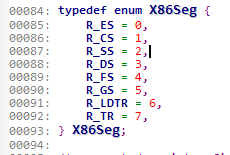


图12

我们看到当前虚拟机断在物理内存0xcb018的位置，看看反汇编如下

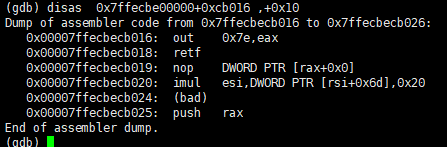


图13

虚拟机中的代码由于out指令导致vm exit，QEMU进行处理io的事件就是把bzImage加载到内存里。

## 问题分析

QEMU中内置了gdb stub。用于支持gdb调试器。主要代码放在gdbstub.c。插入断点的功能是在gdb\_breakpoint\_insert()函数中实现的。

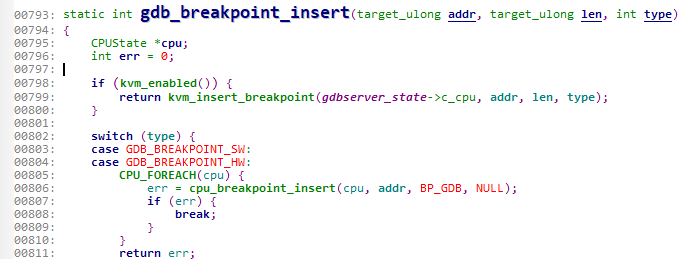


图14

这个问题出在开启KVM,是进入799行的kvm\_insert\_breakpoint（）

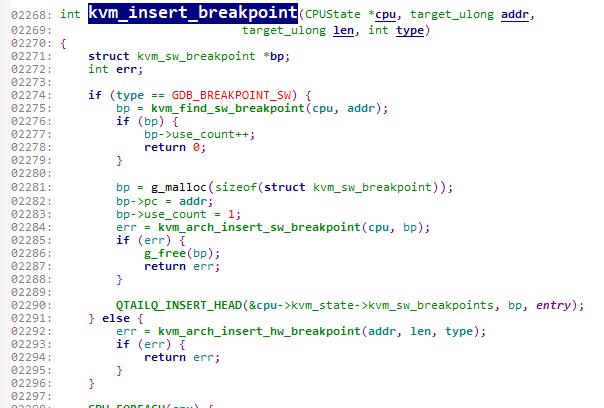


图15

插入的是软断点，进入2284行的kvm\_arch\_insert\_sw\_breakpoint()函数

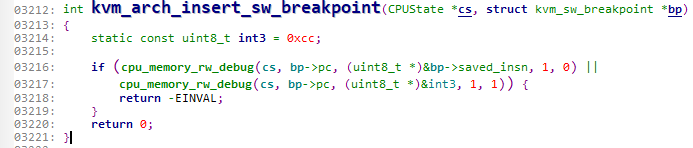


图16

进行跟进cpu\_memory\_rw\_debug()函数

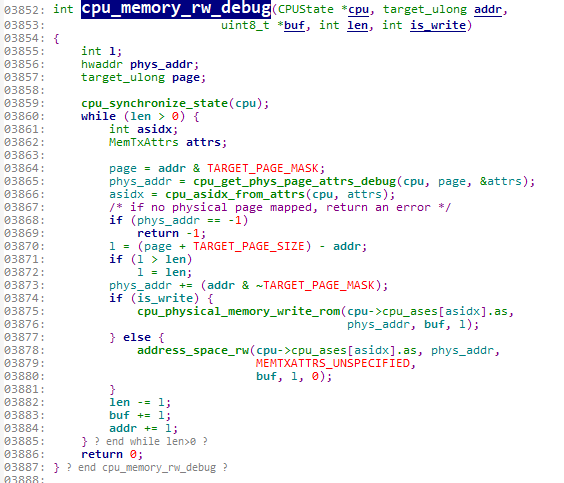


图17

从上面的代码看出KVM下QEMU软断点的方法是先将要设置断点的虚拟地址换成物理地址对应3865行，然后在物理地址下写入0xCC。问题就是出在这。QEMU调试内核内核时加入-s –S 参数后，QEMU会先停下来等gdb的连接，但是此时在前面的背景知识知道这个时候Linux kernel并没有在内存里，而且此时cpu也没有开启分页机制。QEMU会直接向虚拟地址相同的物理位置写入0xCC。

下面是gdb对Linux kernel 内核start\_kernel 下的断点。



图18

QEMU在下面断下来

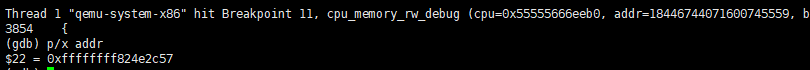


图19



图20

此时由于cpu还没有运行加之分页也没有开启，导致获取的phys\_addr错误，下的断点也自然错误。当QEMU启动过后在对start\_kernel下断点可以到

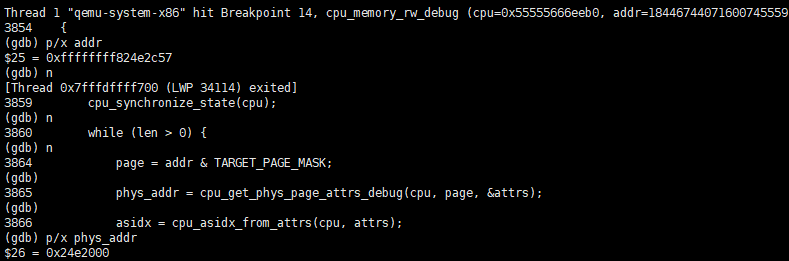


图21

此时工作又正常了。但此时内核已经启动了，在对start\_kernel()下断点也没有用了。所以在QEMU 使用-s –S –enable-kvm这三个参数放在一起时，第一个软件断点不能工作正常。第一个断点可以改用hbreak来解决,后面的断点可以利用软件断点。