使用KGDB调试内核 on QEMU

1: 编译Linux + KGDB

1.1: 安装编译工具

请参考其他相关教程，推荐在安装系统的时候就选择上编译器及支持库

1.2: 下载最新内核代码

在lmkl.org下载最新的内核源码, 可以直接使用浏览器下载代码，当然也可以使用命令wget来下载，下载后需使用tar来解压源代码。这里我们选择linux-2.6.34.1来演示。

假设缺省工作目录为/usr/src/work

```
sudo mkdir -p  /usr/src/work
sudo chmod 777 /usr/src/work -R
cd /usr/src/work
wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.34.1.tar.bz2
tar -jxvf linux-2.6.34.1.tar.bz2
```

1.3: 配置内核选择

使能kgdb调试,并使用kgdboc作为与gdb通信模块。

```
cd linux-2.6.34.1
make defconfig
make menuconfig
```

需确保如下选项被选中(参考kgdb内核选项配置)

```
 General setup  --->
     [ * ] Prompt for development and/or incomplete code/drivers
Kernel hacking  --->
     [ * ] Compile the kernel with debug info
     [ * ] Compile the kernel with frame pointers
     [ * ] KGDB: kernel debugger  --->
           < * >   KGDB: use kgdb over the serial console
```

1.4: 编译

```
make
```

如果你的机器是多核的，可以使用-j+CPU数来进行并行编译，从而加快编译速度，如：

```
make -j2
```

编译完成后，复制bzImage和vmlinux到工作目录下备用

```
cp arch/x86/boot/bzImage /usr/src/work
cp vmlinux /usr/src/work
```

2: 制作自己的文件系统

2.1: 下载busybox

去**busybox站点**下载一个busybox源码包，并解压。

```
cd /usr/src/work
wget http://www.busybox.net/downloads/busybox-1.17.0.tar.bz2
tar -jxvf busybox-1.17.0.tar.bz2
cd busybox-1.17.0
```

2.2: 编译busybox

```
make menuconfig

Busybox Settings  --->
   Build Options  --->
        [ * ] Build BusyBox as a static binary (no shared libs)
   Installation Options  --->
        [ * ] Don't use /usr
Miscellaneous Utilities  --->
[ ] flashcp
[ ] flash_lock
[ ] flash_unlock
[ ] flash_eraseall

注:[ ] 表示不选择
```

保存配置文件后开始编译和安装

```
make
make install
```

此时在当前目录下生成了一个_install目录，里面就是busybox的执行文件

busybox 1.21.0 版本编译问题:

1 找不到pmap_set,pmap_unset引用

禁用neetworking utilities/inetd/support rpc services

2.3: 制作文件系统

使用如下命令来创建一个虚拟文件系统磁盘文件，

在当前目录下创建一个名为busybox.img,大小为100M的文件，并将其格式化为ext3的文件系统

```
cd /usr/src/work
dd if=/dev/zero of=./busybox.img bs=1M count=100
mkfs.ext3 busybox.img
```

将这个虚拟磁盘文件到本地系统中，这样我们可以像访问本地文件一样访问它，
并将生成好的busybox的文件拷贝到这个文件里。

```
sudo mkdir /mnt/disk
sudo mount -o loop /usr/src/work/busybox.img /mnt/disk
sudo cp -rf /usr/src/work/busybox-1.17.0/_install/* /mnt/disk
```

```
创建必须的文件系统目录
cd /mnt/disk/
sudo mkdir dev sys proc etc lib mnt

使用busybox默认的设置文件
sudo cp -a /usr/src/work/busybox-1.17.0/examples/bootfloppy/etc/* /mnt/disk/etc
sudo vi /mnt/disk/etc/init.d/rcS
```

将下面内容拷贝到rcS里:

```
#! /bin/sh
/bin/mount -a
/bin/mount -t  sysfs sysfs /sys
/bin/mount -t tmpfs tmpfs /dev
#动态添加虚拟机环境中的设备
/sbin/mdev -s
```

做完上面对工作后，我们就可以卸载虚拟磁盘文件了。

```
cd /usr/src/work
sudo umount /mnt/disk
```

3: 安装qemu

```
Ubuntu/Debian:
sudo apt-get install qemu

Fedora:
sudo yum install qemu
```

4: 使用qemu运行自己编译的内核

```
 qemu -kernel /usr/src/work/bzImage -append "root=/dev/hda" -boot c -hda
/usr/src/work/busybox.img -k en-us

note:
如果你的硬盘是sata接口的，你也许需要将上面的 "root=/dev/hda" 替换为 "root=/dev/sda".
```

如果顺利的话，自己编译的内核+文件系统就会在那qemu黑乎乎的窗口里展现出来，给自己倒杯水，庆祝下吧！

Qemu在ubuntu上提示加载kvm失败:

1 保证kvm已安装

apt-get install qemu-kvm

2 保证模块已加载

lsmod | grep kvm

3 加载CPU相关的kvm

modprobe kvm-intel

5: gdb + kgdb 调试内核

使能kgdb可以在内核启动时增加使能参数，也可以在内核启动后echo kgdboc模块的参数来达到目的，这里我们采取在内核启动时增加启动参数(kgdboc=ttyS0,115200 kgdbwait)的方式：

```
qemu -kernel /usr/src/work/bzImage -append "root=/dev/hda kgdboc=ttyS0,115200
kgdbwait" -boot c -hda /usr/src/work/busybox.img -k en-us -serial tcp::4321,server
```

这时，运行qemu的终端将提示等待远程连接到本地端口4321：
QEMU waiting for connection on: tcp:0.0.0.0:4321,server

这时使用另外一个控制台执行：

```
gdb /usr/src/work/vmlinux
(gdb) target remote localhost:4321
```

然后qemu就可以继续正常运行下去，最后停止内核，并显示如下信息:
kgdb: Waiting for connection from remote gdb...

这时gdb这边就可以看到如下的提示：

```
(gdb) target remote localhost:4321
Remote debugging using localhost:4321
kgdb_breakpoint () at kernel/debug/debug_core.c:983
983             wmb(); /* Sync point after breakpoint */
(gdb)
```

开始你的内核之旅吧～～～

如果gdb提示如下信息：
warning: Invalid remote reply:
可以使用Ctrl+C来终止当前gdb的操作，再次使用下面命令重新连接一次kgdb即可：
(gdb) target remote localhost:4321

6: gdb + kgdb 调试内核操作示例

待完善...

7: 引用/扩展阅读:

1： 使用 KGDB 调试 Linux 内核（on qemu）
该文章对qemu和kgdb的一些参数介绍得非常详细，本文的4和5章节都是参考它写的，强烈推荐大家看看。
2： setting up kgdb using kvmqemu
该文章也是关于qemu和kgdb，不过它介绍qemu如何联网等内容。PS：和上面的文章不同的是，它将qemu的虚拟串口导向到本地的一个”pty”设备上，
而前面我们是导向到一个socket端口上. qemu -serial参数介绍如下:

```
-serial dev
Redirect the virtual serial port to host character device dev. The default device
is "vc" in graphical mode and "stdio" in non graphical mode. This option can be
used several times to simulate up to 4 serials ports.
```

用DDD+GDB+QEMU 进行linux内核源码级调试

在看此文章之前，请先参看我写的一篇《利用busybox制作一个小巧的linux系统》制作出一个initrd内存盘。

1. 首先编译内核,编译内核时注意要选中

kernel hacking –> kernel debugging –> compile the kernel with debug info

kernel hacking –> compile the kernel with frame pointers

两个选项。(注意:除此之外kernel hacking 选项下其他的选项都不要选,否则会出现断点无法拦截的情况。)

2. 假设你的内核目录位于/home/xxx/linux-2.6.28中，且编译内核时你指定的大O参数指定的目录是O=/home/xxx/linux-2.6.28-obj。

用新编译的带调试信息的内核，启动一个虚拟机。

$ qemu –kernel /home/xxx/linux-2.6.28-obj/arch/x86/boot/bzImage –append "root=/dev/ram0 rw"

-initrd /home/initrd.gz –s –S 具体选项的含义可以参看qemu的文档。

这样启动虚拟机后，它会在1234端口产生一个gdb stub以供调试时使用。

然后我们打开令一个xterm终端运行gdb命令开始调试：

$ gdb /home/xxx/linux-2.6.28-obj/vmlinux

然后在gdb的提示符下输入: target remote localhost:1234 连接gdb stub

然后开始设置断点,比如: break start_kernel

然后输入c命令，然后qemu继续运行后，就会在start_kernel 入口处停了下来。

这时候你就可以输入各种gdb命令来对linux kernel进行hack了。。

当然，这里采用gdb的一个前端工具ddd会更好。但还是需要学习一些常用的gdb命令(这样才能更熟练的进行调试)

常用的gdb调试命令:

1. file <文件名>： 加载被调试的可执行程序文件.

2. run(简写r也可以): 运行被调试程序,直到遇到断点.

3. c : 继续执行被调试程序,直到下一断点.

4. b : 设置断点.

5. d <编号>：删除断点.

6. info breakpoints : 显示已设置的断点列表.

7. s : 源码级的单步进入.

8. n : 源码级的单步步过.

9. si/ni : 指令级的单步进入和步过.(需要先运行display /i $pc)

10. info all-register: 显示所有寄存器的值.

11. p /x $eax : 以16进制显示某一特定寄存器的值.

12. q: 退出gdb调试环境.

13. disassemble <0xXXXXXXXX>：反汇编指定地址处的指令.(加/r选项,显示对应汇编代码的机器码)

14. set disassembly-flavor <intel | att>：设定汇编显示的格式(Intel格式或者AT&T格式)

15. 修改指定地址的内存数据:set {unsigned char} <0xXXXXXXXX> = <new data>

16. 显示源代码文件: list

17. info stack: 查看栈追踪(栈回溯). 或者使用 backtrace 也可以

18. x /nfu <addr> : 查看指定地址处的内存内容.

n: 表示个数. f(format): x,16进制. d,10进制. i,指令. u(每一项的长度): b,单字节. w,4字节. h,双字节.

例如: x /1xw 0x804a010 : 显示0x804a010起始地址处的一个4字节的16进制值.

x /100i 0x804a010 : 显示0x804010起始地址往后的100个指令