LINUX内核开发与调试 -- KGDB

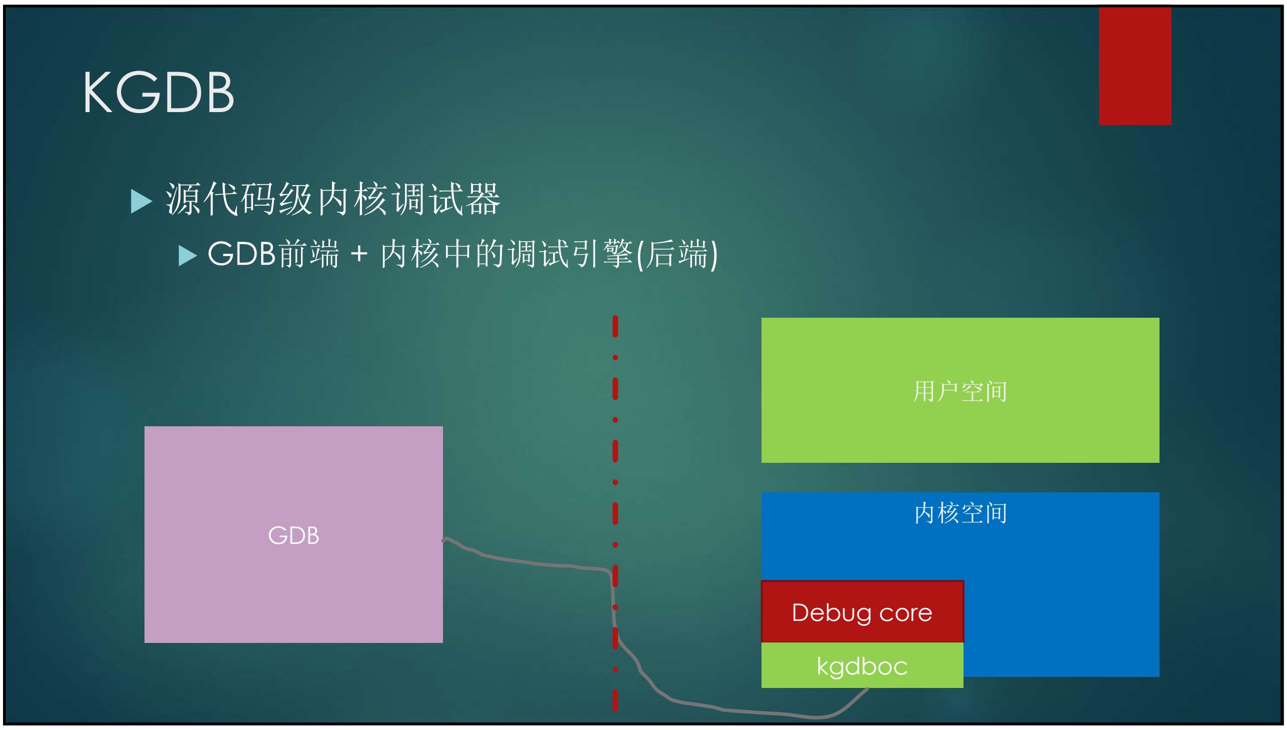
张银奎

2017/5/1

1

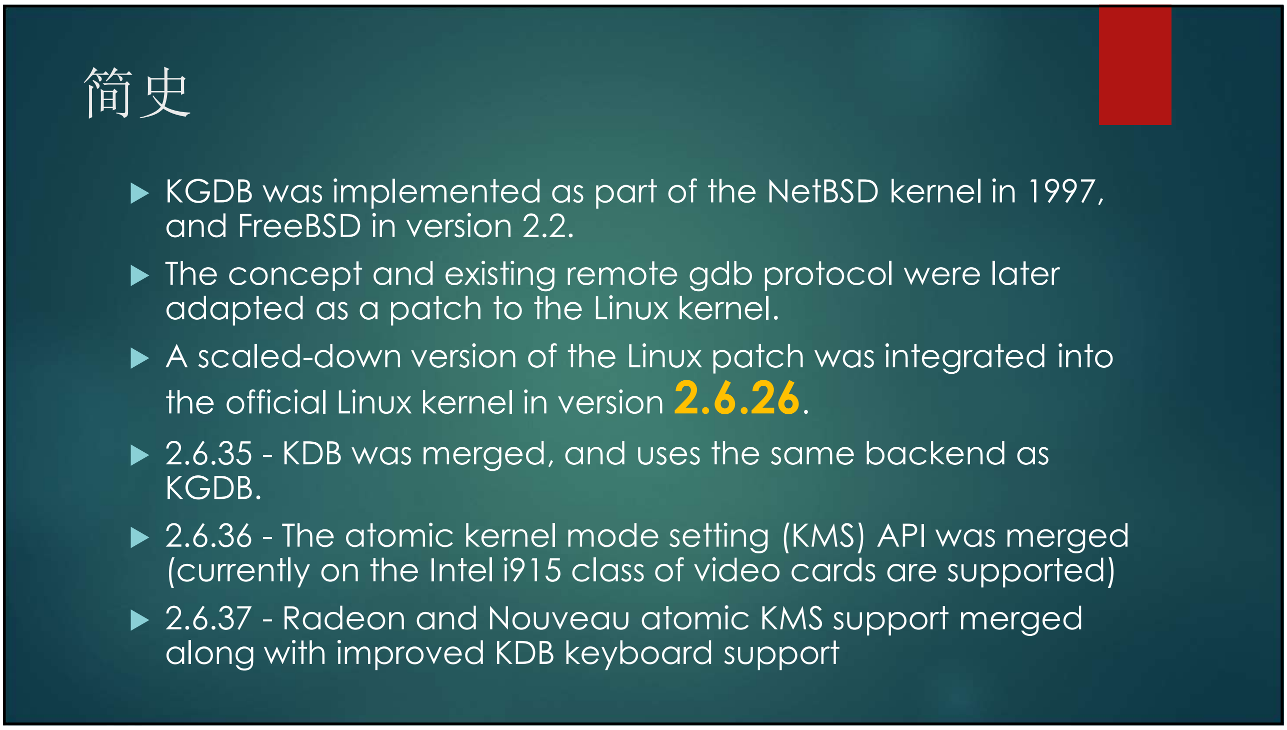
概览 → 准备符号文件 → 启用和连接 → 双机调试 → kcore

2



KGDB

- 源代码级内核调试器
  - GDB前端 + 内核中的调试引擎(后端)



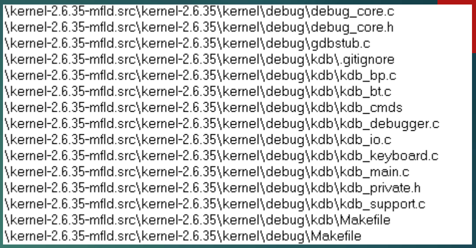
简史

- KGDB was implemented as part of the NetBSD kernel in 1997, and FreeBSD in version 2.2.
- The concept and existing remote gdb protocol were later adapted as a patch to the Linux kernel.
- A scaled-down version of the Linux patch was integrated into the official Linux kernel in version **2.6.26**.
- 2.6.35 - KDB was merged, and uses the same backend as KGDB.
- 2.6.36 - The atomic kernel mode setting (KMS) API was merged (currently on the Intel i915 class of video cards are supported)
- 2.6.37 - Radeon and Nouveau atomic KMS support merged along with improved KDB keyboard support

Re: Availability of kdb

- Date: Wed, 6 Sep 2000 12:52:29 -0700 (PDT)
- From: Linus Torvalds
- “I don't like debuggers. Never have, probably never will.”
- “So I don‘t make it part of the standard distribution…”

2.6.26



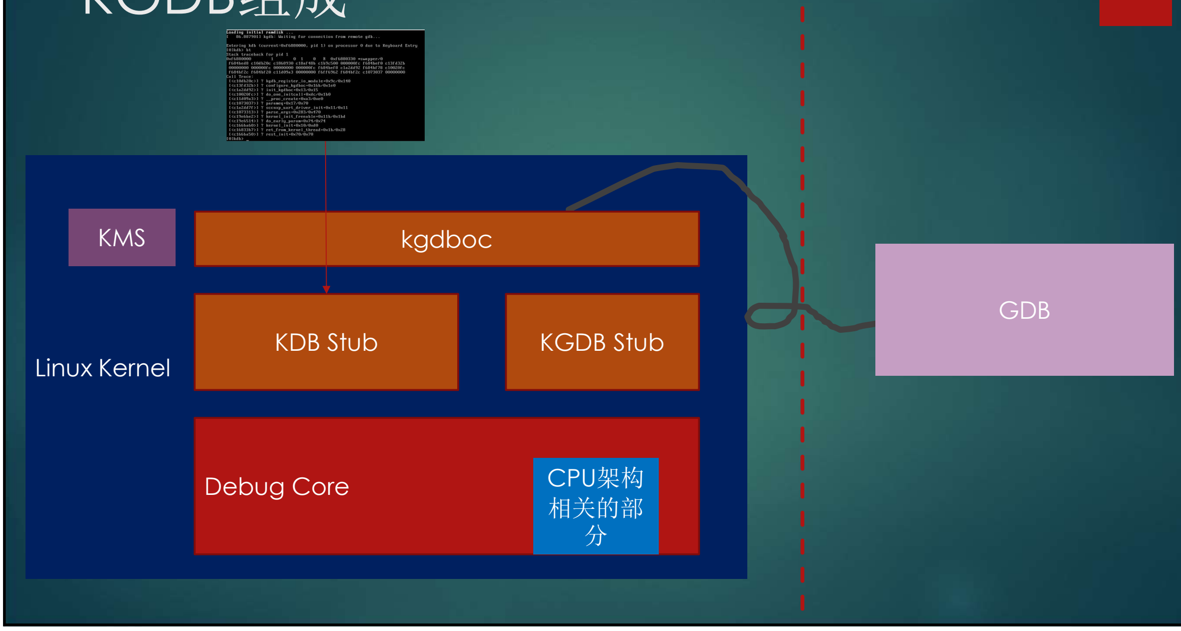
```
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\debug_core.c
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\debug_core.h
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\gdbstub.c
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\kdb\.gitignore
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\kdb\kdb_bp.c
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\kdb\kdb_bt.c
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\kdb\kdb_cmds
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\kdb\kdb_debugger.c
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\kdb\kdb_io.c
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\kdb\kdb_keyboard.c
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\kdb\kdb_main.c
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\kdb\kdb_private.h
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\kdb\kdb_support.c
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\kdb\Makefile
\kernel-2.6.35-mfld.src\kernel-2.6.35\kernel\debug\Makefile
```

- **Subject**: Linux 2.6.26-rc1
- **Date**: Sat, 3 May 2008 12:38:20 -0700 (PDT)
- “Another feature that is notable not for its size, but because people have tried to get me to merge it for some long is kgdb support. Which really turned out pretty small and clean, once people started putting their effort into making it so.”

维护者

- Amit Kale maintained the Linux KGDB from 2000 to 2004.
- From 2004 to 2006, it was maintained by Linsyssoft Technologies, after which Jason Wessel at Wind River Systems, Inc. took over as the official maintainer.
- Ingo Molnar and Jason Wessel created a slimmed-down and cleaned up version of KGDB which was called "kgdb light" (without Ethernet support and many other hacks). This was the one merged into the 2.6.26 kernel. This version of kgdb supports only RS-232 connectivity, using a special driver which can split debugger inputs and console inputs such that only a single serial port is required.

KGDB组成



```
Loading initial ramdisk ...
[   86.887901] kgdb: Waiting for connection from remote gdb...

Entering kdb (current=0xf6880000, pid 1) on processor 0 due to Keyboard Entry
[0]kdb> bt
Stack traceback for pid 1
0xf6880000        1        0  1    0   R  0xf6880330 *swapper/0
 f684bed8 c10db20c c1868930 c18af48b c1b9c500 000000fc f684bef0 c13fd32b
 00000000 000000fc 00000000 000000fc f684bef8 c1a2dd92 f684bf78 c10020fc
 f684bf2c f684bf20 c11d09a3 00000000 f6ff6962 f684bf2c c1073037 00000000
Call Trace:
 [<c10db20c>] ? kgdb_register_io_module+0x9c/0x140
 [<c13fd32b>] ? configure_kgdboc+0x16b/0x1e0
 [<c1a2dd92>] ? init_kgdboc+0x13/0x15
 [<c10020fc>] ? do_one_initcall+0xdc/0x1b0
 [<c11d09a3>] ? __proc_create+0xa3/0xe0
 [<c1073037>] ? parameq+0x17/0x70
 [<c1a2dd7f>] ? sccnxp_uart_driver_init+0x11/0x11
 [<c1073313>] ? parse_args+0x283/0x470
 [<c19e6be2>] ? kernel_init_freeable+0x11b/0x1bd
 [<c19e6514>] ? do_early_param+0x74/0x74
 [<c166ba60>] ? kernel_init+0x10/0xd0
 [<c16833b7>] ? ret_from_kernel_thread+0x1b/0x28
 [<c166ba50>] ? rest_init+0x70/0x70
[0]kdb> _
```

KMS

kgdboc

Linux Kernel

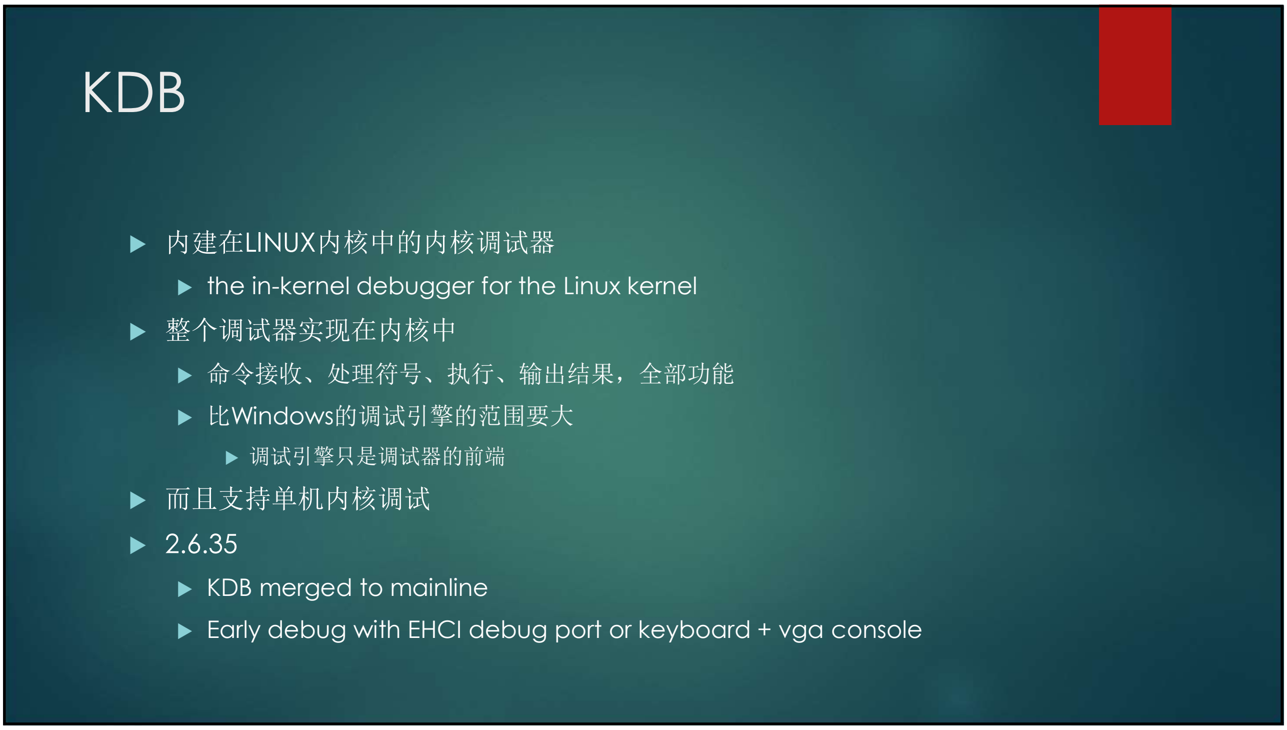
KDB Stub

KGDB Stub

Debug Core

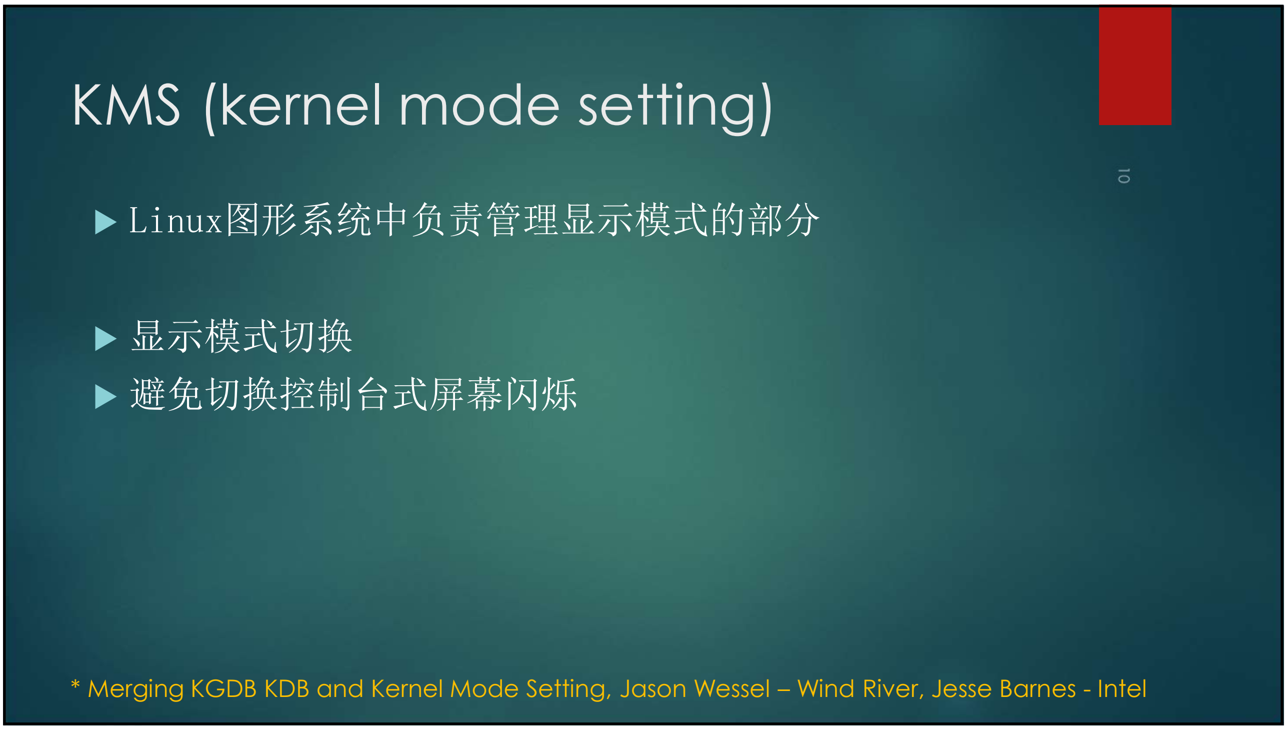
CPU架构相关的部分

GDB



KDB

- 内建在LINUX内核中的内核调试器
  - the in-kernel debugger for the Linux kernel
- 整个调试器实现在内核中
  - 命令接收、处理符号、执行、输出结果，全部功能
  - 比Windows的调试引擎的范围要大
    - 调试引擎只是调试器的前端
- 而且支持单机内核调试
- 2.6.35
  - KDB merged to mainline
  - Early debug with EHCI debug port or keyboard + vga console

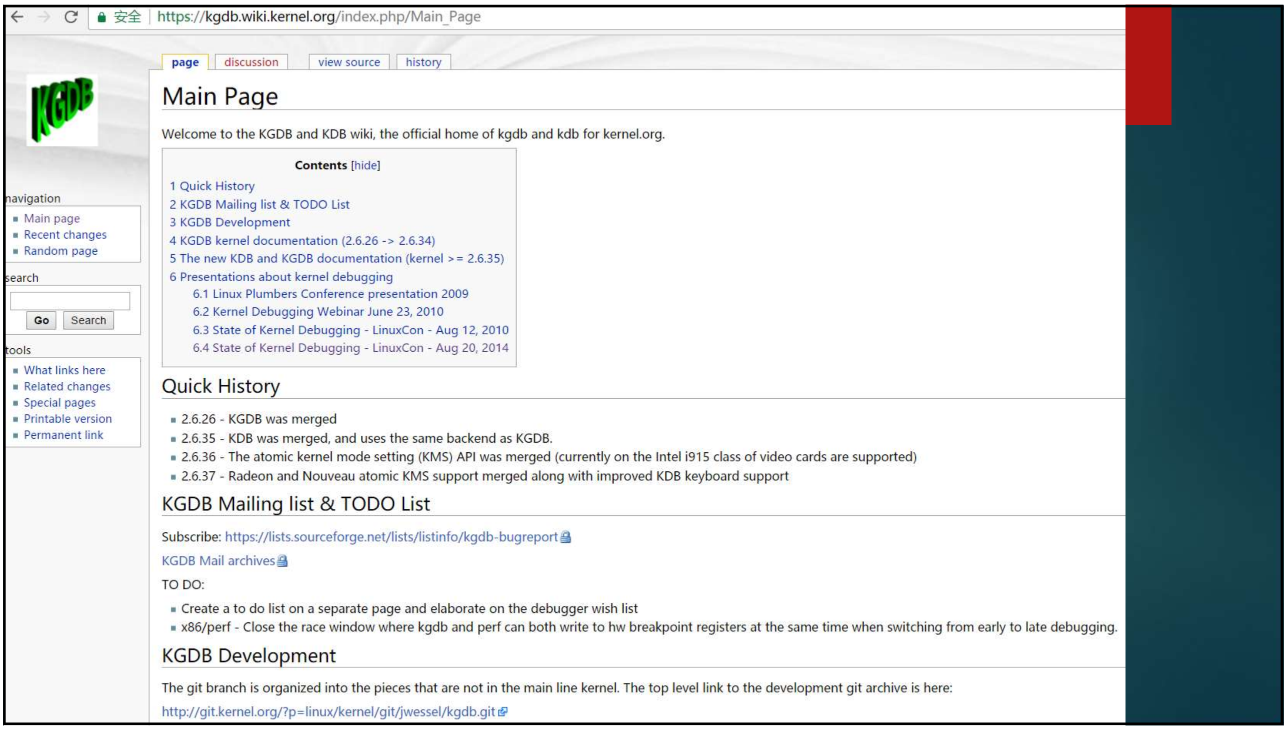


KMS (kernel mode setting)

- Linux图形系统中负责管理显示模式的部分

- 显示模式切换
- 避免切换控制台式屏幕闪烁

* Merging KGDB KDB and Kernel Mode Setting, Jason Wessel – Wind River, Jesse Barnes - Intel



Main Page

Welcome to the KGDB and KDB wiki, the official home of kgdb and kdb for kernel.org.

Contents [hide]

Quick History

- 2.6.26 - KGDB was merged
- 2.6.35 - KDB was merged, and uses the same backend as KGDB.
- 2.6.36 - The atomic kernel mode setting (KMS) API was merged (currently on the Intel i915 class of video cards are supported)
- 2.6.37 - Radeon and Nouveau atomic KMS support merged along with improved KDB keyboard support

KGDB Mailing list & TODO List

Subscribe: https://lists.sourceforge.net/lists/listinfo/kgdb-bugreport

KGDB Mail archives

TO DO:

- Create a to do list on a separate page and elaborate on the debugger wish list
- x86/perf - Close the race window where kgdb and perf can both write to hw breakpoint registers at the same time when switching from early to late debugging.

KGDB Development

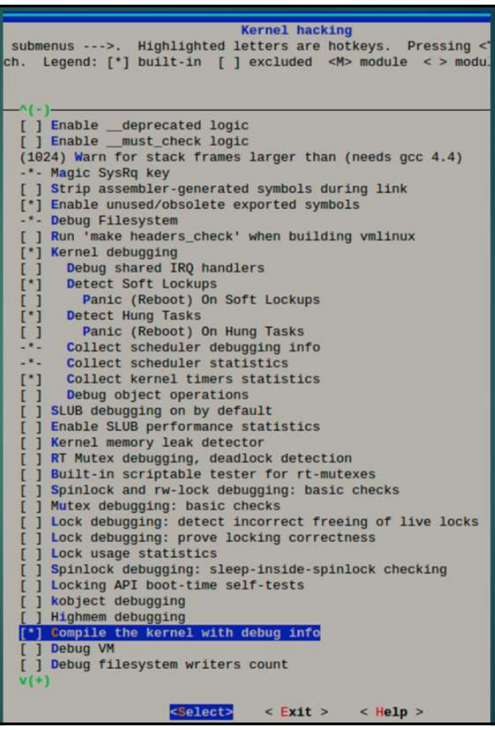
The git branch is organized into the pieces that are not in the main line kernel. The top level link to the development git archive is here:

http://git.kernel.org/?p=linux/kernel/git/jwessel/kgdb.git

概览 → 准备符号文件 → 启用和连接 → 双机调试 → kcore

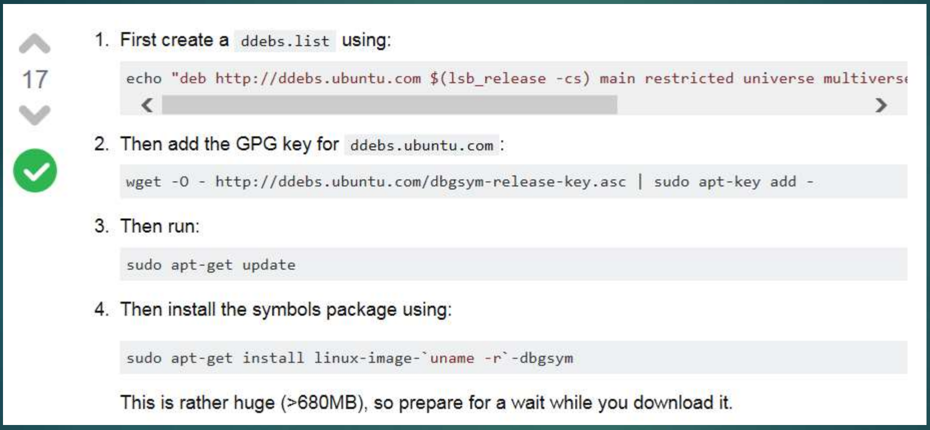
调试信息

- 源代码跟踪
- 局部变量
- 类型信息



```
                    Kernel hacking
 submenus --->.  Highlighted letters are hotkeys.  Pressing <
ch.  Legend: [*] built-in  [ ] excluded  <M> module  < > modu

 ^(-)
  [ ] Enable __deprecated logic
  [ ] Enable __must_check logic
  (1024) Warn for stack frames larger than (needs gcc 4.4)
  -*- Magic SysRq key
  [ ] Strip assembler-generated symbols during link
  [*] Enable unused/obsolete exported symbols
  -*- Debug Filesystem
  [ ] Run 'make headers_check' when building vmlinux
  [*] Kernel debugging
  [ ]   Debug shared IRQ handlers
  [*]   Detect Soft Lockups
  [ ]     Panic (Reboot) On Soft Lockups
  [*]   Detect Hung Tasks
  [ ]     Panic (Reboot) On Hung Tasks
  -*-   Collect scheduler debugging info
  -*-   Collect scheduler statistics
  [*]   Collect kernel timers statistics
  [ ]   Debug object operations
  [ ] SLUB debugging on by default
  [ ] Enable SLUB performance statistics
  [ ] Kernel memory leak detector
  [ ] RT Mutex debugging, deadlock detection
  [ ] Built-in scriptable tester for rt-mutexes
  [ ] Spinlock and rw-lock debugging: basic checks
  [ ] Mutex debugging: basic checks
  [ ] Lock debugging: detect incorrect freeing of live locks
  [ ] Lock debugging: prove locking correctness
  [ ] Lock usage statistics
  [ ] Spinlock debugging: sleep-inside-spinlock checking
  [ ] Locking API boot-time self-tests
  [ ] kobject debugging
  [ ] Highmem debugging
  [*] Compile the kernel with debug info
  [ ] Debug VM
  [ ] Debug filesystem writers count
  v(+)

            <Select>    < Exit >    < Help >
```

下载内核符号文件



1. First create a `ddebs.list` using:

```
echo "deb http://ddebs.ubuntu.com $(lsb_release -cs) main restricted universe multiverse
```

2. Then add the GPG key for `ddebs.ubuntu.com`:

```
wget -O - http://ddebs.ubuntu.com/dbgsym-release-key.asc | sudo apt-key add -
```

3. Then run:

```
sudo apt-get update
```

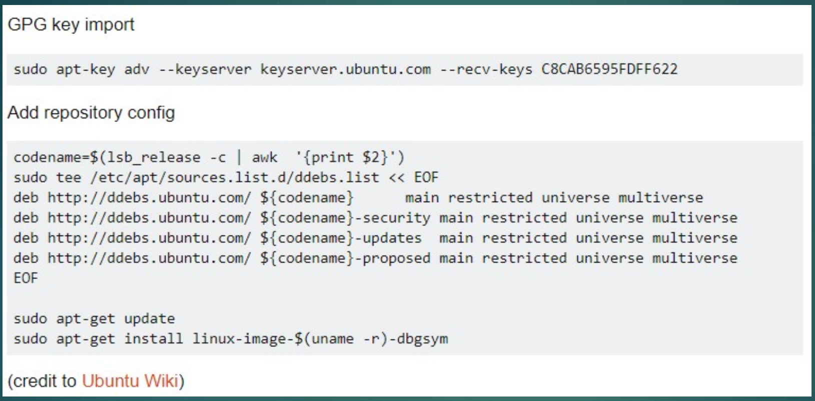
4. Then install the symbols package using:

```
sudo apt-get install linux-image-`uname -r`-dbgsym
```

This is rather huge (>680MB), so prepare for a wait while you download it.

http://askubuntu.com/questions/197016/how-to-install-a-package-that-contains-ubuntu-kernel-debug-symbols

16.04+



GPG key import

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys C8CAB6595FDFF622
```

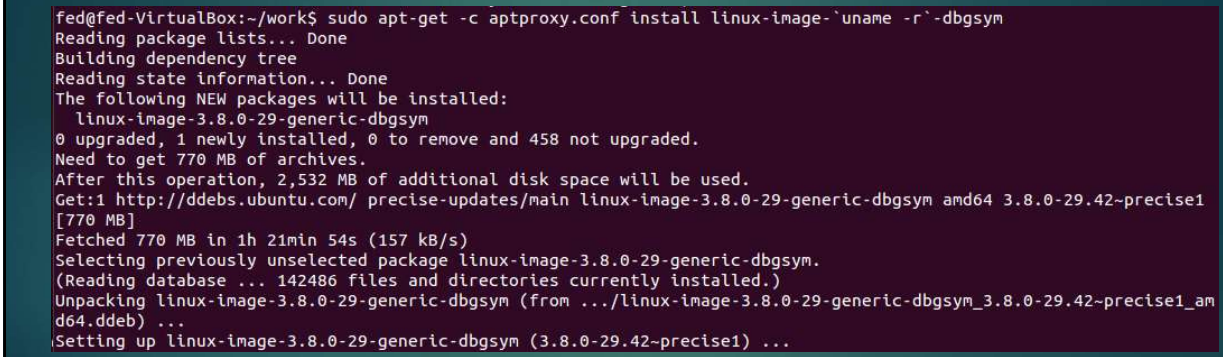
Add repository config

```
codename=$(lsb_release -c | awk  '{print $2}')
sudo tee /etc/apt/sources.list.d/ddebs.list << EOF
deb http://ddebs.ubuntu.com/ ${codename}      main restricted universe multiverse
deb http://ddebs.ubuntu.com/ ${codename}-security main restricted universe multiverse
deb http://ddebs.ubuntu.com/ ${codename}-updates  main restricted universe multiverse
deb http://ddebs.ubuntu.com/ ${codename}-proposed main restricted universe multiverse
EOF

sudo apt-get update
sudo apt-get install linux-image-$(uname -r)-dbgsym
```

(credit to Ubuntu Wiki)

- https://askubuntu.com/questions/197016/how-to-install-a-package-that-contains-ubuntu-kernel-debug-symbols

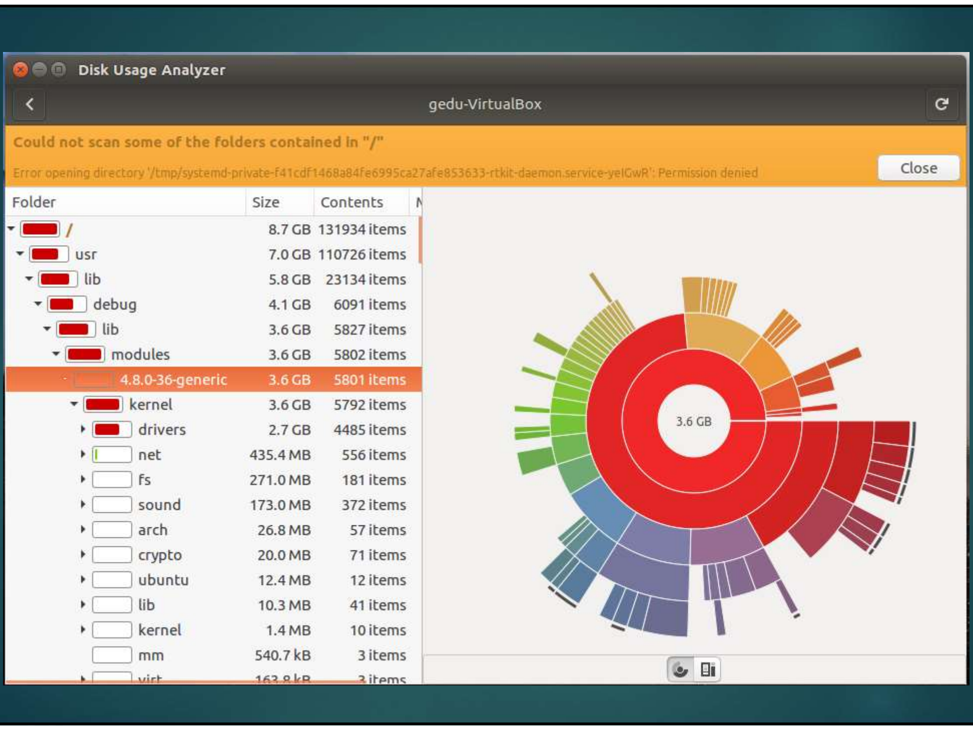
示例



```
fed@fed-VirtualBox:~/work$ sudo apt-get -c aptproxy.conf install linux-image-`uname -r`-dbgsym
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  linux-image-3.8.0-29-generic-dbgsym
0 upgraded, 1 newly installed, 0 to remove and 458 not upgraded.
Need to get 770 MB of archives.
After this operation, 2,532 MB of additional disk space will be used.
Get:1 http://ddebs.ubuntu.com/ precise-updates/main linux-image-3.8.0-29-generic-dbgsym amd64 3.8.0-29.42~precise1
[770 MB]
Fetched 770 MB in 1h 21min 54s (157 kB/s)
Selecting previously unselected package linux-image-3.8.0-29-generic-dbgsym.
(Reading database ... 142486 files and directories currently installed.)
Unpacking linux-image-3.8.0-29-generic-dbgsym (from .../linux-image-3.8.0-29-generic-dbgsym_3.8.0-29.42~precise1_am
d64.ddeb) ...
Setting up linux-image-3.8.0-29-generic-dbgsym (3.8.0-29.42~precise1) ...
```

老雷博客

- http://advdbg.org/blogs/advdbg_system/articles/7147.aspx



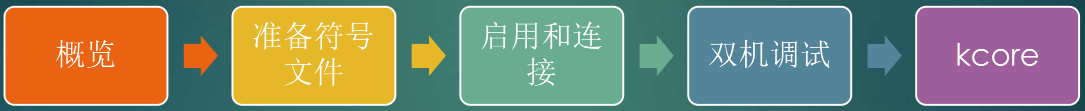
Disk Usage Analyzer

gedu-VirtualBox

Could not scan some of the folders contained in "/"

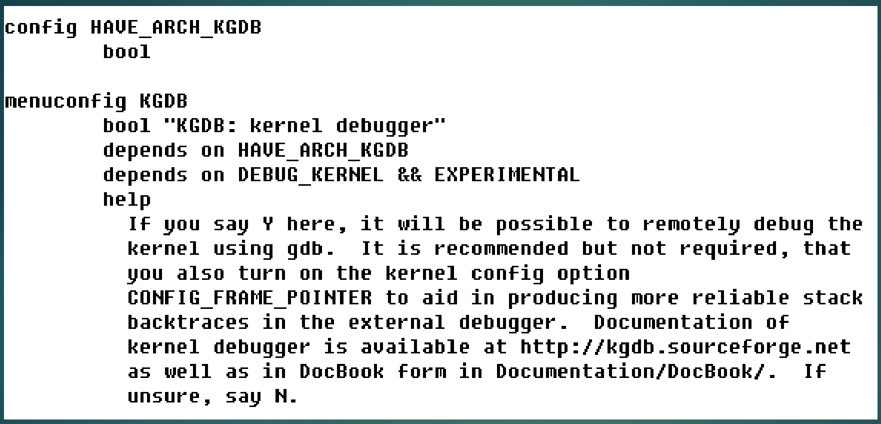
Error opening directory '/tmp/systemd-private-f41cdf1468a84fe6995ca27afe853633-rtkit-daemon.service-yelGwR': Permission denied

Folder	Size	Contents
/	8.7 GB	131934 items
usr	7.0 GB	110726 items
lib	5.8 GB	23134 items
debug	4.1 GB	6091 items
lib	3.6 GB	5827 items
modules	3.6 GB	5802 items
4.8.0-36-generic	3.6 GB	5801 items
kernel	3.6 GB	5792 items
drivers	2.7 GB	4485 items
net	435.4 MB	556 items
fs	271.0 MB	181 items
sound	173.0 MB	372 items
arch	26.8 MB	57 items
crypto	20.0 MB	71 items
ubuntu	12.4 MB	12 items
lib	10.3 MB	41 items
kernel	1.4 MB	10 items
mm	540.7 kB	3 items
virt	163.8 kB	3 items



概览 → 准备符号文件 → 启用和连接 → 双机调试 → kcore

编译时启用



```
config HAVE_ARCH_KGDB
        bool

menuconfig KGDB
        bool "KGDB: kernel debugger"
        depends on HAVE_ARCH_KGDB
        depends on DEBUG_KERNEL && EXPERIMENTAL
        help
          If you say Y here, it will be possible to remotely debug the
          kernel using gdb.  It is recommended but not required, that
          you also turn on the kernel config option
          CONFIG_FRAME_POINTER to aid in producing more reliable stack
          backtraces in the external debugger.  Documentation of
          kernel debugger is available at http://kgdb.sourceforge.net
          as well as in DocBook form in Documentation/DocBook/.  If
          unsure, say N.
```

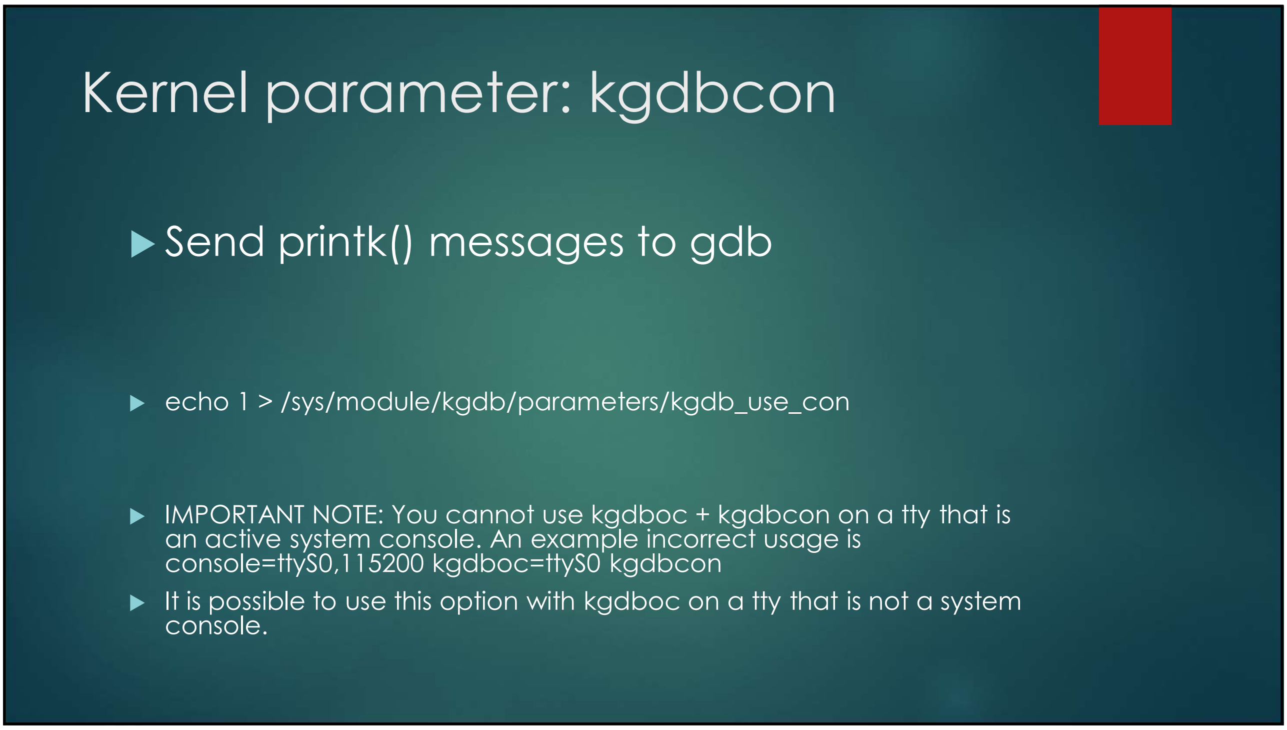
- http://kernel.org/pub/linux/kernel/people/jwessel/kdb/
- https://www.ridgerun.com/developer/wiki/index.php/How_to_use_kgdb

kernel configuration

- CONFIG_DEBUG_INFO is in the Kernel hacking | Compile-time checks and compiler options | Compile the kernel with debug info menu
- CONFIG_FRAME_POINTER may be an option for your architecture, and is in the Kernel hacking | Compile-time checks and compiler options | Compile the kernel with frame pointers menu
- CONFIG_KGDB is in the Kernel hacking | KGDB: kernel debugger menu
- CONFIG_KGDB_SERIAL_CONSOLE is in the Kernel hacking | KGDB: kernel debugger | KGDB: use kgdb over the serial console menu

40-custom

```
menuentry 'Ubuntu, with Linux 3.12.2 Kernel Debug Serial' --class ubuntu --class gnu-linux --class gnu -
-class os {
    recordfail
    gfxmode $linux_gfx_mode
    insmod gzio
    insmod part_msdos
    insmod ext2
    set root='(hd0,msdos1)'
    search --no-floppy --fs-uuid --set=root eb5d8aee-6a0f-4257-b5b8-8d6731ba6764
    echo 'Loading Linux 3.12.2 with KGDB built by GEDU lab...'
    linux    /boot/vmlinuz-3.12.2 root=UUID=eb5d8aee-6a0f-4257-b5b8-8d6731ba6764 ro quiet
splash nomodeset $vt_handoff print-fatal-signals=1 kgdbwait
kgdb8250=io,03f8,ttyS0,115200,4 kgdboc=ttyS0,115200
kgdboe=@192.168.26.244/,@192.168.26.245/ kgdbcon nokaslr
    echo 'Loading initial ramdisk ...'
    initrd    /boot/initrd.img-3.12.2
}
```

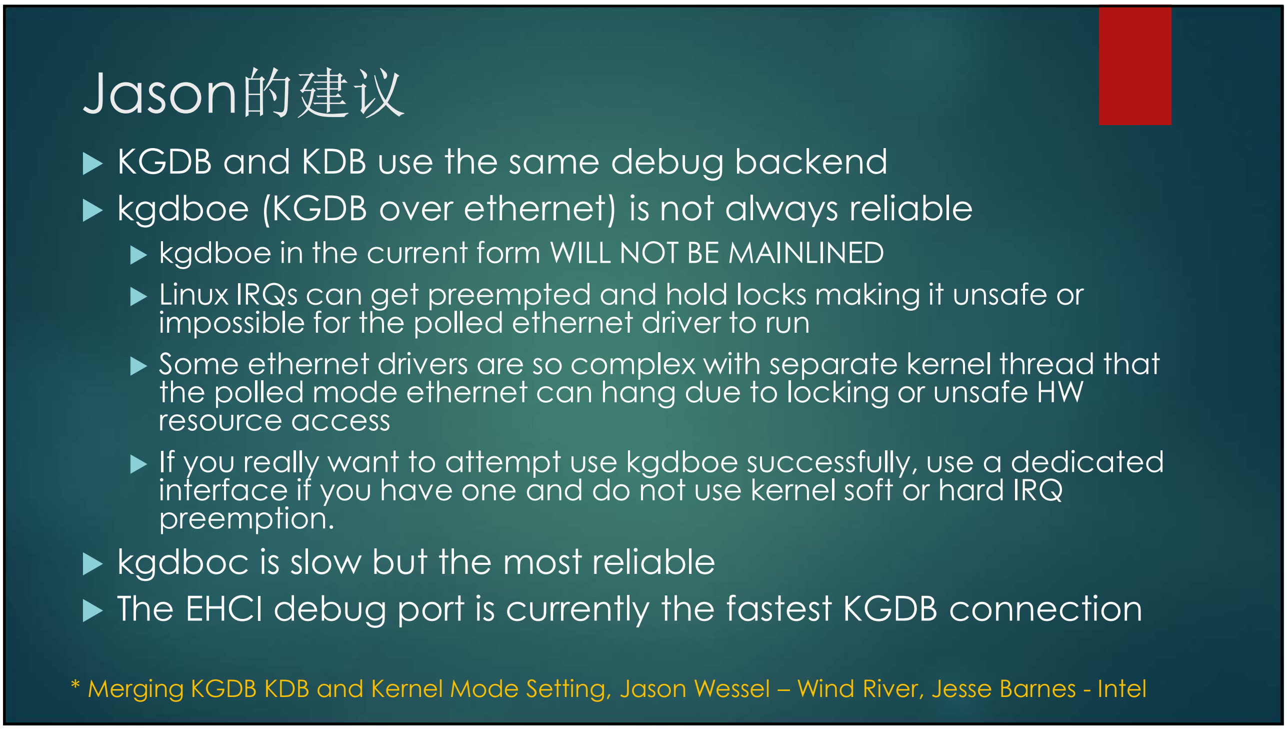


Kernel parameter: kgdbcon

- Send printk() messages to gdb

- echo 1 > /sys/module/kgdb/parameters/kgdb_use_con

- IMPORTANT NOTE: You cannot use kgdboc + kgdbcon on a tty that is an active system console. An example incorrect usage is console=ttyS0,115200 kgdboc=ttyS0 kgdbcon
- It is possible to use this option with kgdboc on a tty that is not a system console.

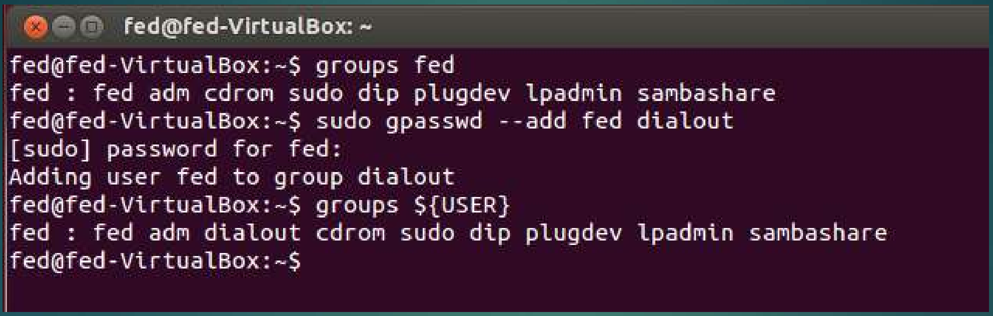


Jason的建议

- KGDB and KDB use the same debug backend
- kgdboe (KGDB over ethernet) is not always reliable
  - kgdboe in the current form WILL NOT BE MAINLINED
  - Linux IRQs can get preempted and hold locks making it unsafe or impossible for the polled ethernet driver to run
  - Some ethernet drivers are so complex with separate kernel thread that the polled mode ethernet can hang due to locking or unsafe HW resource access
  - If you really want to attempt use kgdboe successfully, use a dedicated interface if you have one and do not use kernel soft or hard IRQ preemption.
- kgdboc is slow but the most reliable
- The EHCI debug port is currently the fastest KGDB connection

* Merging KGDB KDB and Kernel Mode Setting, Jason Wessel – Wind River, Jesse Barnes - Intel

测试串口通信



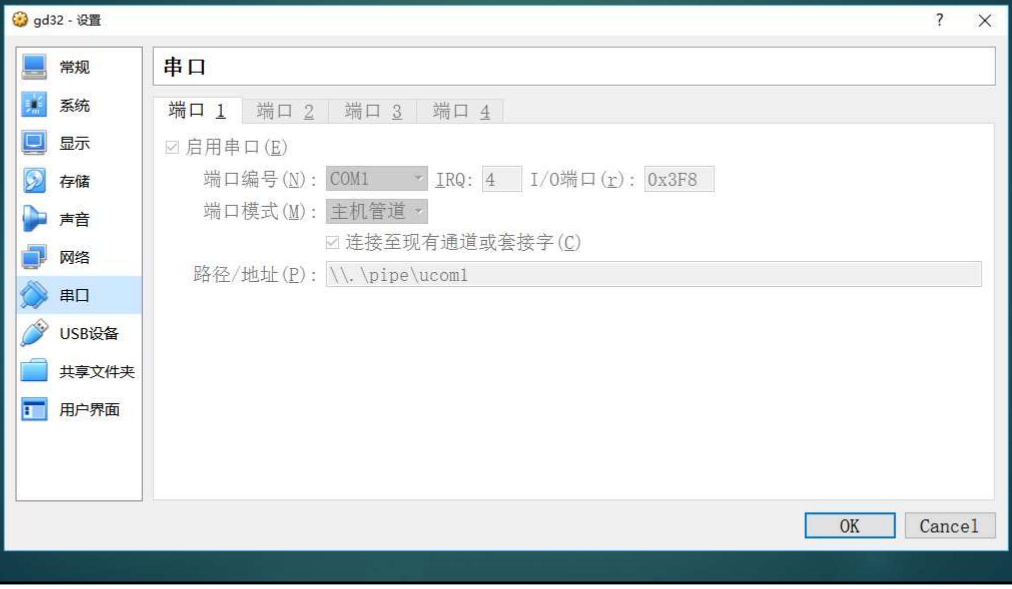
```
fed@fed-VirtualBox: ~
fed@fed-VirtualBox:~$ groups fed
fed : fed adm cdrom sudo dip plugdev lpadmin sambashare
fed@fed-VirtualBox:~$ sudo gpasswd --add fed dialout
[sudo] password for fed:
Adding user fed to group dialout
fed@fed-VirtualBox:~$ groups ${USER}
fed : fed adm dialout cdrom sudo dip plugdev lpadmin sambashare
fed@fed-VirtualBox:~$
```

The only solution that works for me is to: (every time I boot the machine)

sudo chmod 666 /dev/ttys0
It really needs to be fixed at time of installation. I'm on 15.10 and have tried 16.04 LTS, still the same there. Seems like such a simple fix.

The older versions 10.04LTS did not have this problem.

虚拟机



串口

端口 1　端口 2　端口 3　端口 4

☑ 启用串口(E)

端口编号(N)：COM1　IRQ: 4　I/O端口(r)：0x3F8

端口模式(M)：主机管道

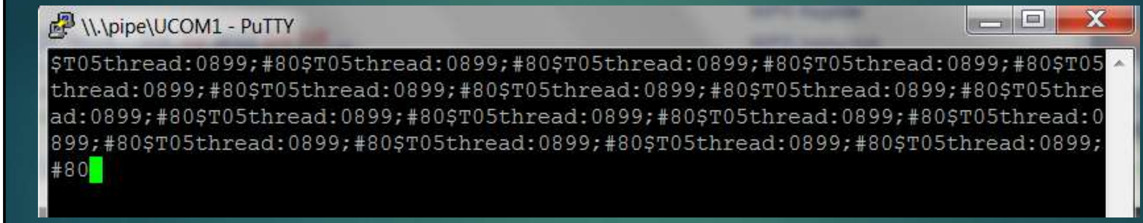
☑ 连接至现有通道或套接字(C)

路径/地址(P)：\\.\pipe\ucom1

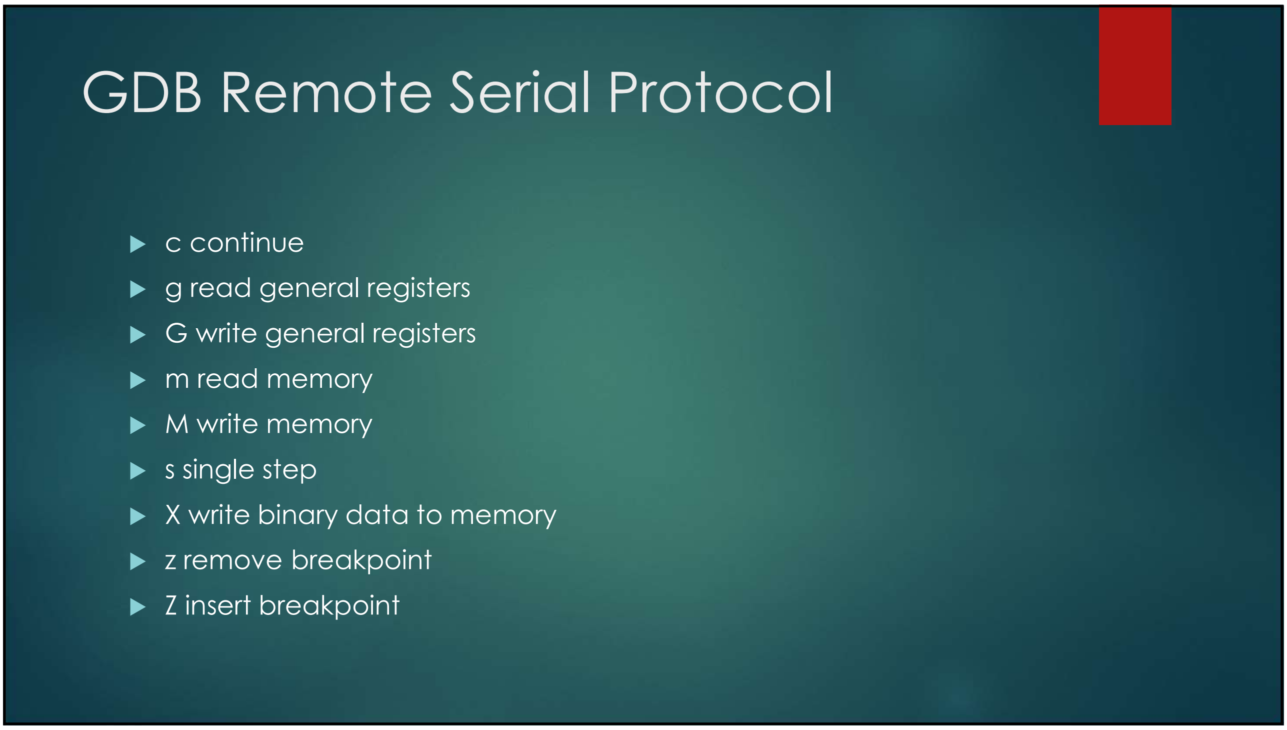
主机端

- % cd /home/ge/work/linux-3.12.2
- % sudo gdb -s vmlinux
- (gdb) set remotebaud 115200
- (gdb) target remote /dev/ttyS0

通信数据

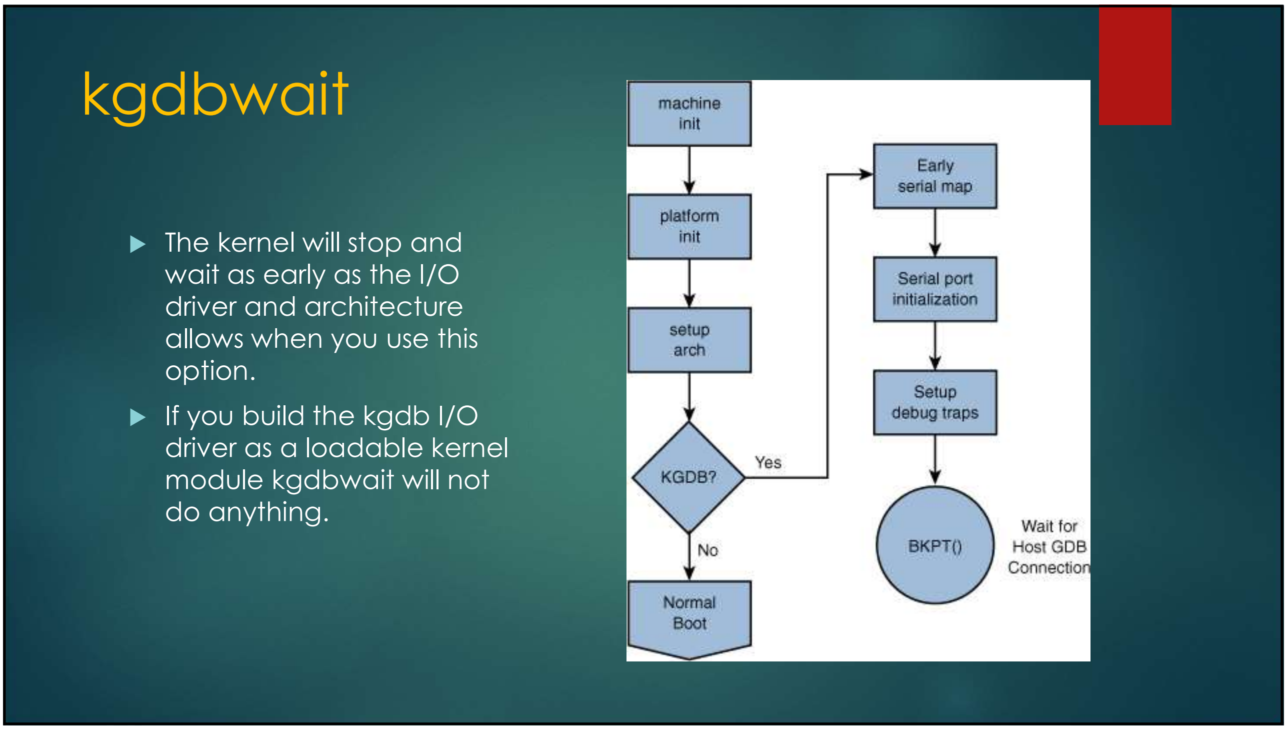


```
$T05thread:0899;#80$T05thread:0899;#80$T05thread:0899;#80$T05thread:0899;#80$T05
thread:0899;#80$T05thread:0899;#80$T05thread:0899;#80$T05thread:0899;#80$T05thre
ad:0899;#80$T05thread:0899;#80$T05thread:0899;#80$T05thread:0899;#80$T05thread:0
899;#80$T05thread:0899;#80$T05thread:0899;#80$T05thread:0899;#80$T05thread:0899;
#80
```



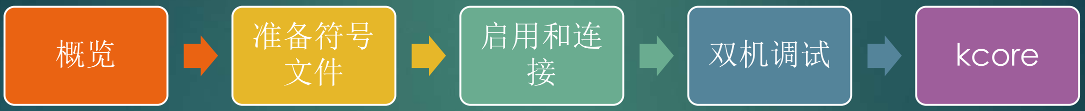
GDB Remote Serial Protocol

- c continue
- g read general registers
- G write general registers
- m read memory
- M write memory
- s single step
- X write binary data to memory
- z remove breakpoint
- Z insert breakpoint



kgdbwait

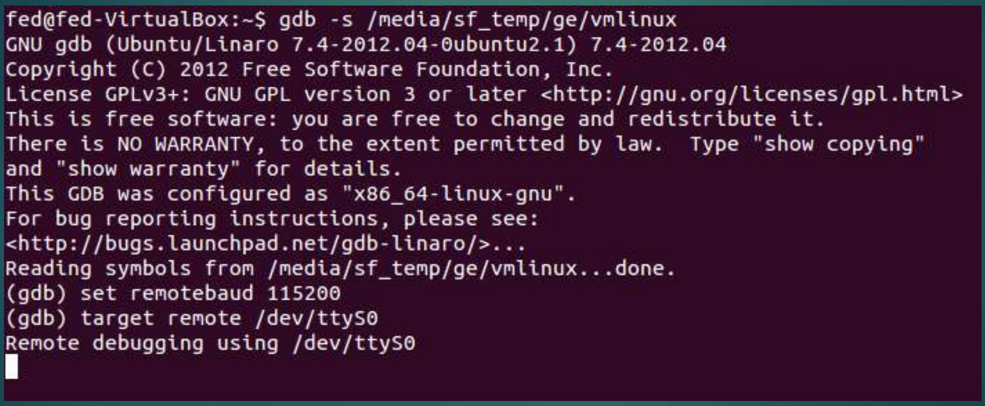
- The kernel will stop and wait as early as the I/O driver and architecture allows when you use this option.
- If you build the kgdb I/O driver as a loadable kernel module kgdbwait will not do anything.



概览 → 准备符号文件 → 启用和连接 → 双机调试 → kcore

31

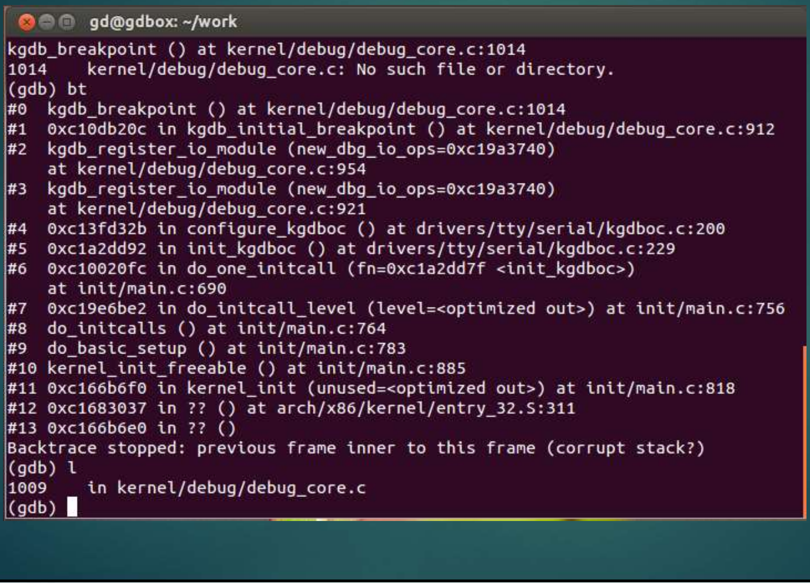
启动GDB



```
fed@fed-VirtualBox:~$ gdb -s /media/sf_temp/ge/vmlinux
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 "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /media/sf_temp/ge/vmlinux...done.
(gdb) set remotebaud 115200
(gdb) target remote /dev/ttyS0
Remote debugging using /dev/ttyS0
█
```

32

初始断点



```
kgdb_breakpoint () at kernel/debug/debug_core.c:1014
1014    kernel/debug/debug_core.c: No such file or directory.
(gdb) bt
#0  kgdb_breakpoint () at kernel/debug/debug_core.c:1014
#1  0xc10db20c in kgdb_initial_breakpoint () at kernel/debug/debug_core.c:912
#2  kgdb_register_io_module (new_dbg_io_ops=0xc19a3740)
    at kernel/debug/debug_core.c:954
#3  kgdb_register_io_module (new_dbg_io_ops=0xc19a3740)
    at kernel/debug/debug_core.c:921
#4  0xc13fd32b in configure_kgdboc () at drivers/tty/serial/kgdboc.c:200
#5  0xc1a2dd92 in init_kgdboc () at drivers/tty/serial/kgdboc.c:229
#6  0xc10020fc in do_one_initcall (fn=0xc1a2dd7f <init_kgdboc>)
    at init/main.c:690
#7  0xc19e6be2 in do_initcall_level (level=<optimized out>) at init/main.c:756
#8  do_initcalls () at init/main.c:764
#9  do_basic_setup () at init/main.c:783
#10 kernel_init_freeable () at init/main.c:885
#11 0xc166b6f0 in kernel_init (unused=<optimized out>) at init/main.c:818
#12 0xc1683037 in ?? () at arch/x86/kernel/entry_32.S:311
#13 0xc166b6e0 in ?? ()
Backtrace stopped: previous frame inner to this frame (corrupt stack?)
(gdb) l
1009    in kernel/debug/debug_core.c
(gdb)
```

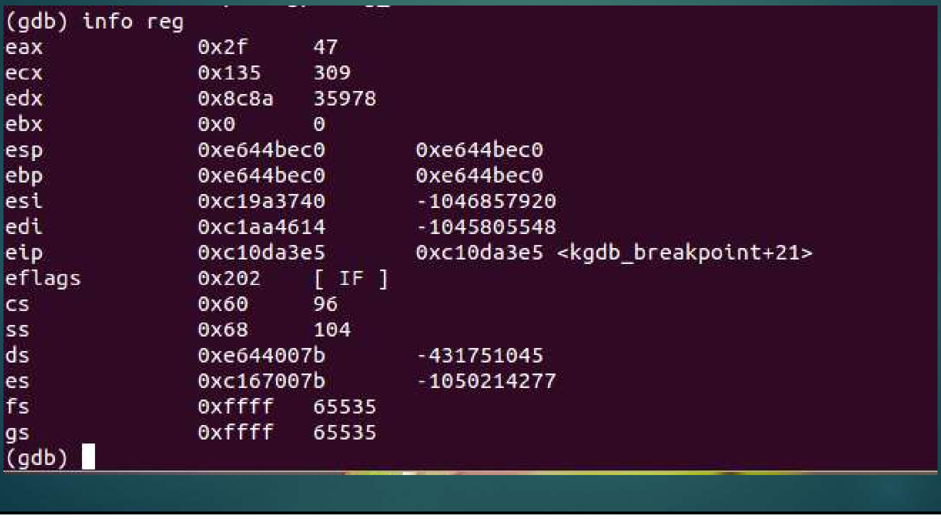
First Break

```
(gdb) bt
#0  kgdb_breakpoint () at kernel/debug/debug_core.c:1014
#1  0xc10db20c in kgdb_initial_breakpoint ()
    at kernel/debug/debug_core.c:912
#2  kgdb_register_io_module (new_dbg_io_ops=0xc19a3740)
    at kernel/debug/debug_core.c:954
#3  kgdb_register_io_module (new_dbg_io_ops=0xc19a3740)
    at kernel/debug/debug_core.c:921
#4  0xc13fd32b in configure_kgdboc ()
    at drivers/tty/serial/kgdboc.c:200
#5  0xc1a2dd92 in init_kgdboc ()
    at drivers/tty/serial/kgdboc.c:229
#6  0xc10020fc in do_one_initcall (
    fn=0xc1a2dd7f <init_kgdboc>) at init/main.c:690
#7  0xc19e6be2 in do_initcall_level (level=<optimized out>)
    at init/main.c:756
#8  do_initcalls () at init/main.c:764
#9  do_basic_setup () at init/main.c:783
#10 kernel_init_freeable () at init/main.c:885
#11 0xc166b6f0 in kernel_init (unused=<optimized out>)
    at init/main.c:818
#12 0xc1683037 in ?? () at arch/x86/kernel/entry_32.S:311
#13 0xc166b6e0 in ?? ()
Backtrace stopped: previous frame inner to this frame (corrupt ---Type <return> to continue, or q <return> to q
stack?)
```

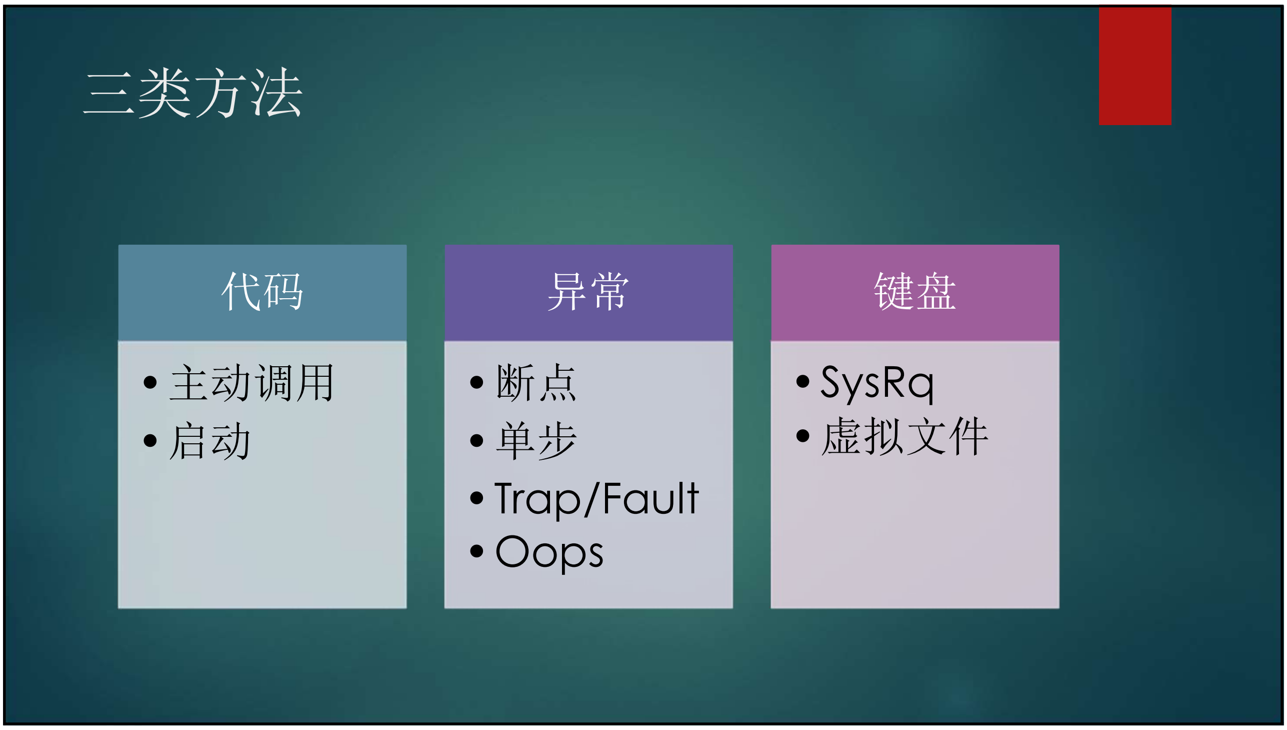
Debugging early code

- kgdboc=ttyO0,115200 kgdbwait
- Now, when you boot, you will see this on the console:
- 1.103415] console [ttyO0] enabled
- [1.108216] kgdb: Registered I/O driver kgdboc.
- [1.113071] kgdb: Waiting for connection from remote gdb...
- At this point, you can close the console and connect from GDB in the usual way.

观察寄存器



```
(gdb) info reg
eax            0x2f     47
ecx            0x135    309
edx            0x8c8a   35978
ebx            0x0      0
esp            0xe644bec0       0xe644bec0
ebp            0xe644bec0       0xe644bec0
esi            0xc19a3740       -1046857920
edi            0xc1aa4614       -1045805548
eip            0xc10da3e5       0xc10da3e5 <kgdb_breakpoint+21>
eflags         0x202    [ IF ]
cs             0x60     96
ss             0x68     104
ds             0xe644007b       -431751045
es             0xc167007b       -1050214277
fs             0xffff   65535
gs             0xffff   65535
(gdb) 
```



三类方法

代码

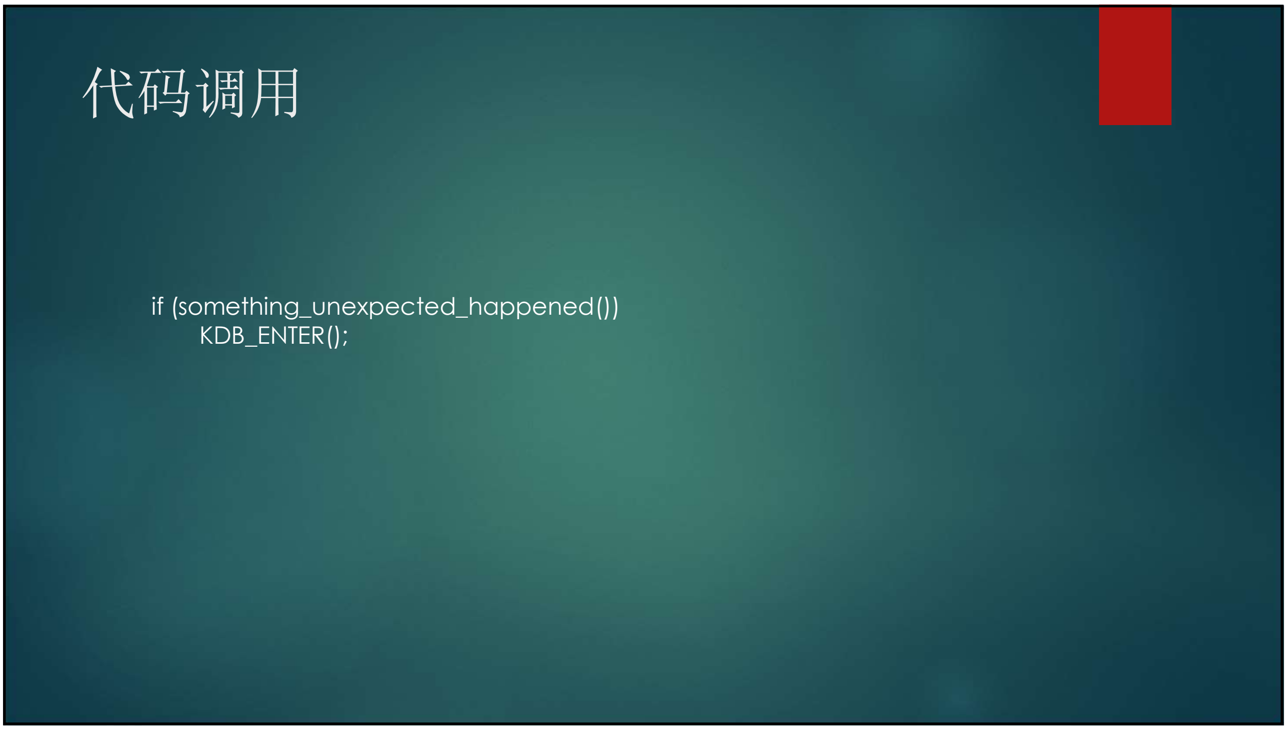
- 主动调用
- 启动

异常

- 断点
- 单步
- Trap/Fault
- Oops

键盘

- SysRq
- 虚拟文件

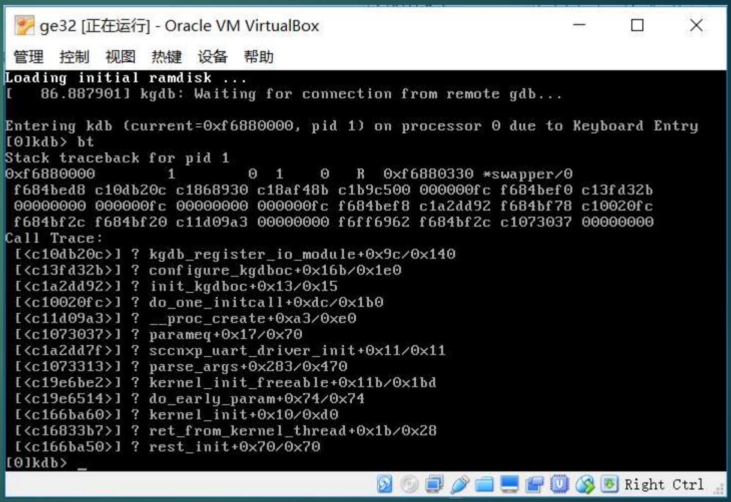


代码调用

```
if (something_unexpected_happened())
        KDB_ENTER();
```

初始断点

```
// kgdb_register_io_module
// debug_core.c

if (kgdb_break_asap)
        kgdb_initial_breakpoint();
```


ge32 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Right Ctrl

```
Loading initial ramdisk ...
[   86.887901] kgdb: Waiting for connection from remote gdb...

Entering kdb (current=0xf6880000, pid 1) on processor 0 due to Keyboard Entry
[0]kdb> bt
Stack traceback for pid 1
0xf6880000        1         0  1    0   R  0xf6880330 *swapper/0
 f684bed8 c10db20c c1868930 c18af48b c1b9c500 000000fc f684bef0 c13fd32b
 00000000 000000fc 00000000 000000fc f684bef8 c1a2dd92 f684bf78 c10020fc
 f684bf2c f684bf20 c11d09a3 00000000 f6ff6962 f684bf2c c1073037 00000000
Call Trace:
 [<c10db20c>] ? kgdb_register_io_module+0x9c/0x140
 [<c13fd32b>] ? configure_kgdboc+0x16b/0x1e0
 [<c1a2dd92>] ? init_kgdboc+0x13/0x15
 [<c10020fc>] ? do_one_initcall+0xdc/0x1b0
 [<c11d09a3>] ? __proc_create+0xa3/0xe0
 [<c1073037>] ? parameq+0x17/0x70
 [<c1a2dd7f>] ? sccnxp_uart_driver_init+0x11/0x11
 [<c1073313>] ? parse_args+0x283/0x470
 [<c19e6be2>] ? kernel_init_freeable+0x11b/0x1bd
 [<c19e6514>] ? do_early_param+0x74/0x74
 [<c166ba60>] ? kernel_init+0x10/0xd0
 [<c16833b7>] ? ret_from_kernel_thread+0x1b/0x28
 [<c166ba50>] ? rest_init+0x70/0x70
[0]kdb> _
```

Model F 'AT'

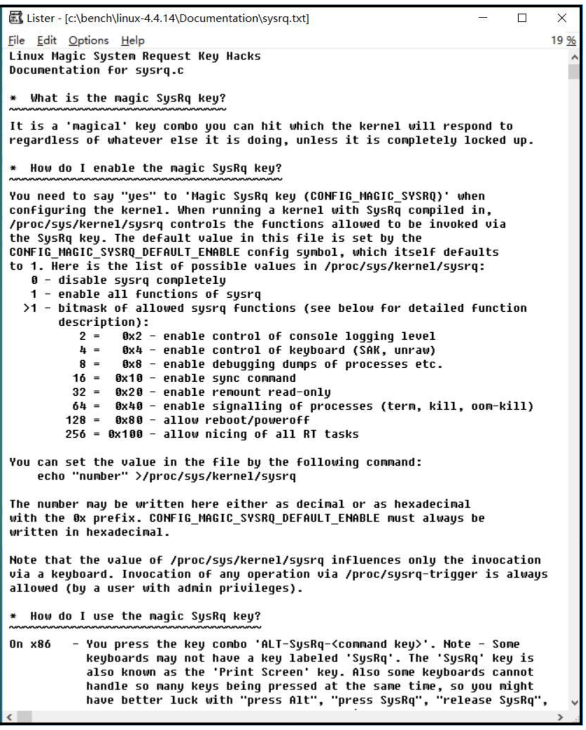
SysReq

按法

On x86 - You press the key combo 'ALT-SysRq-<command key>'. Note - Some keyboards may not have a key labeled 'SysRq'. The 'SysRq' key is also known as the 'Print Screen' key. Also some keyboards cannot handle so many keys being pressed at the same time, so you might have better luck with "press Alt", "press SysRq", "release SysRq","press <command key>", release everything.

*** 对老雷使用的HP Inspiration 7000应该使用后一种方法
可能禁止，使用如下命令启用
echo 1 > /proc/sys/kernel/sysrq

Ubuntu 16.04默认禁止g功能
/etc/sysctl.d/10-magic-sysrq.conf中配置为176
$ echo 1 > /proc/sys/kernel/sysrq 后即可以



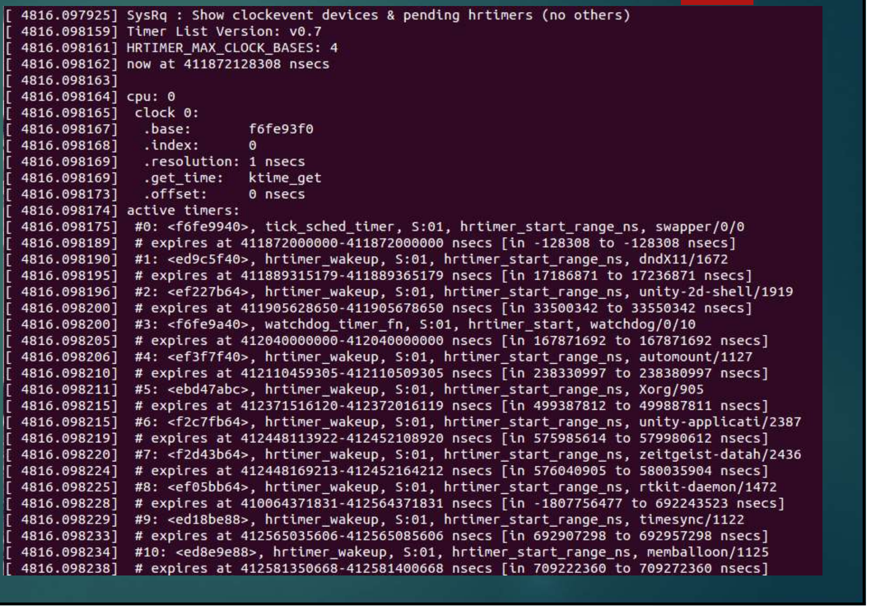
```
Linux Magic System Request Key Hacks
Documentation for sysrq.c

*  What is the magic SysRq key?
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

It is a 'magical' key combo you can hit which the kernel will respond to
regardless of whatever else it is doing, unless it is completely locked up.

*  How do I enable the magic SysRq key?
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

You need to say "yes" to 'Magic SysRq key (CONFIG_MAGIC_SYSRQ)' when
configuring the kernel. When running a kernel with SysRq compiled in,
/proc/sys/kernel/sysrq controls the functions allowed to be invoked via
the SysRq key. The default value in this file is set by the
CONFIG_MAGIC_SYSRQ_DEFAULT_ENABLE config symbol, which itself defaults
to 1. Here is the list of possible values in /proc/sys/kernel/sysrq:
   0 - disable sysrq completely
   1 - enable all functions of sysrq
  >1 - bitmask of allowed sysrq functions (see below for detailed function
       description):
          2 =   0x2 - enable control of console logging level
          4 =   0x4 - enable control of keyboard (SAK, unraw)
          8 =   0x8 - enable debugging dumps of processes etc.
         16 =  0x10 - enable sync command
         32 =  0x20 - enable remount read-only
         64 =  0x40 - enable signalling of processes (term, kill, oom-kill)
        128 =  0x80 - allow reboot/poweroff
        256 = 0x100 - allow nicing of all RT tasks

You can set the value in the file by the following command:
    echo "number" >/proc/sys/kernel/sysrq

The number may be written here either as decimal or as hexadecimal
with the 0x prefix. CONFIG_MAGIC_SYSRQ_DEFAULT_ENABLE must always be
written in hexadecimal.

Note that the value of /proc/sys/kernel/sysrq influences only the invocation
via a keyboard. Invocation of any operation via /proc/sysrq-trigger is always
allowed (by a user with admin privileges).

*  How do I use the magic SysRq key?
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

On x86   - You press the key combo 'ALT-SysRq-<command key>'. Note - Some
           keyboards may not have a key labeled 'SysRq'. The 'SysRq' key is
           also known as the 'Print Screen' key. Also some keyboards cannot
           handle so many keys being pressed at the same time, so you might
           have better luck with "press Alt", "press SysRq", "release SysRq",
```

命令键

'b'	Will immediately reboot the system without syncing or unmounting your disks.
'c'	Will perform a system crash by a NULL pointer dereference. A crashdump will be taken if configured.
'd'	Shows all locks that are held.
'e'	Send a SIGTERM to all processes, except for init.
'f'	Will call the oom killer to kill a memory hog process, but do not panic if nothing can be killed.
'g'	Used by kgdb (kernel debugger)
'h'	Will display help (actually any other key than those listed here will display help. but 'h' is easy to remember :-)
'i'	Send a SIGKILL to all processes, except for init.
'j'	Forcibly "Just thaw it" filesystems frozen by the FIFREEZE ioctl.
'k'	Secure Access Key (SAK) Kills all programs on the current virtual console. NOTE: See important comments below in SAK section.
'l'	Shows a stack backtrace for all active CPUs.
'm'	Will dump current memory info to your console.
'n'	Used to make RT tasks nice-able
'o'	Will shut your system off (if configured and supported).
'p'	Will dump the current registers and flags to your console.
'q'	Will dump per CPU lists of all armed hrtimers (but NOT regular timer_list timers) and detailed information about all clockevent devices.
'r'	Turns off keyboard raw mode and sets it to XLATE.
's'	Will attempt to sync all mounted filesystems.
't'	Will dump a list of current tasks and their information to your console.
'u'	Will attempt to remount all mounted filesystems read-only.
'v'	Forcefully restores framebuffer console
'v'	Causes ETM buffer dump [ARM-specific]
'w'	Dumps tasks that are in uninterruptable (blocked) state.
'x'	Used by xmon interface on ppc/powerpc platforms. Show global PMU Registers on sparc64. Dump all TLB entries on MIPS.
'y'	Show global CPU Registers [SPARC-64 specific]
'z'	Dump the ftrace buffer
'0'-'9'	Sets the console log level, controlling which kernel messages will be printed to your console.('0', for example would make it so that only emergency messages like PANICs or OOPSes would make it to your console.)

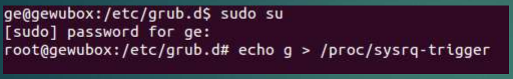
SysRq

[4776.667571] SysRq : HELP : loglevel(0-9) reboot(b) crash(c) terminate-all-tasks(e) memory-full-oom-kill(f) kill-all-tasks(i) thaw-filesystems(j) sak(k) show-backtrace-all-active-cpus(l) show-memory-usage(m) nice-all-RT-tasks(n) poweroff(o) show-registers(p) show-all-timers(q) unraw(r) sync(s) show-task-states(t) unmount(u) show-blocked-tasks(w) dump-ftrace-buffer(z)



```
[ 4816.097925] SysRq : Show clockevent devices & pending hrtimers (no others)
[ 4816.098159] Timer List Version: v0.7
[ 4816.098161] HRTIMER_MAX_CLOCK_BASES: 4
[ 4816.098162] now at 411872128308 nsecs
[ 4816.098163]
[ 4816.098164] cpu: 0
[ 4816.098165]  clock 0:
[ 4816.098167]   .base:       f6fe93f0
[ 4816.098168]   .index:      0
[ 4816.098169]   .resolution: 1 nsecs
[ 4816.098169]   .get_time:   ktime_get
[ 4816.098173]   .offset:     0 nsecs
[ 4816.098174] active timers:
[ 4816.098175]  #0: <f6fe9940>, tick_sched_timer, S:01, hrtimer_start_range_ns, swapper/0/0
[ 4816.098189]  # expires at 411872000000-411872000000 nsecs [in -128308 to -128308 nsecs]
[ 4816.098190]  #1: <ed9c5f40>, hrtimer_wakeup, S:01, hrtimer_start_range_ns, dndX11/1672
[ 4816.098195]  # expires at 411889315179-411889365179 nsecs [in 17186871 to 17236871 nsecs]
[ 4816.098196]  #2: <ef227b64>, hrtimer_wakeup, S:01, hrtimer_start_range_ns, unity-2d-shell/1919
[ 4816.098200]  # expires at 411905628650-411905678650 nsecs [in 33500342 to 33550342 nsecs]
[ 4816.098200]  #3: <f6fe9a40>, watchdog_timer_fn, S:01, hrtimer_start, watchdog/0/10
[ 4816.098205]  # expires at 412040000000-412040000000 nsecs [in 167871692 to 167871692 nsecs]
[ 4816.098206]  #4: <ef3f7f40>, hrtimer_wakeup, S:01, hrtimer_start_range_ns, automount/1127
[ 4816.098210]  # expires at 412110459305-412110509305 nsecs [in 238330997 to 238380997 nsecs]
[ 4816.098211]  #5: <ebd47abc>, hrtimer_wakeup, S:01, hrtimer_start_range_ns, Xorg/905
[ 4816.098215]  # expires at 412371516120-412372016119 nsecs [in 499387812 to 499887811 nsecs]
[ 4816.098215]  #6: <f2c7fb64>, hrtimer_wakeup, S:01, hrtimer_start_range_ns, unity-applicati/2387
[ 4816.098219]  # expires at 412448113922-412452108920 nsecs [in 575985614 to 579980612 nsecs]
[ 4816.098220]  #7: <f2d43b64>, hrtimer_wakeup, S:01, hrtimer_start_range_ns, zeitgeist-datah/2436
[ 4816.098224]  # expires at 412448169213-412452164212 nsecs [in 576040905 to 580035904 nsecs]
[ 4816.098225]  #8: <ef05bb64>, hrtimer_wakeup, S:01, hrtimer_start_range_ns, rtkit-daemon/1472
[ 4816.098228]  # expires at 410064371831-412564371831 nsecs [in -1807756477 to 692243523 nsecs]
[ 4816.098229]  #9: <ed18be88>, hrtimer_wakeup, S:01, hrtimer_start_range_ns, timesync/1122
[ 4816.098233]  # expires at 412565035606-412565085606 nsecs [in 692907298 to 692957298 nsecs]
[ 4816.098234]  #10: <ed8e9e88>, hrtimer_wakeup, S:01, hrtimer_start_range_ns, memballoon/1125
[ 4816.098238]  # expires at 412581350668-412581400668 nsecs [in 709222360 to 709272360 nsecs]
```

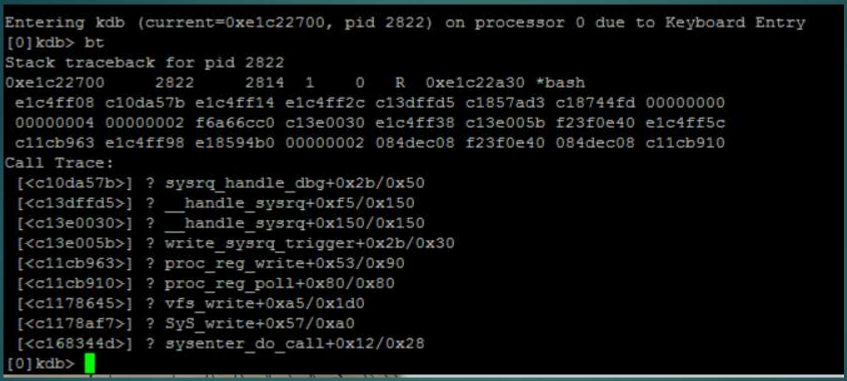
Break-in过程



```
ge@gewubox:/etc/grub.d$ sudo su
[sudo] password for ge:
root@gewubox:/etc/grub.d# echo g > /proc/sysrq-trigger
```

```
#0  kgdb_breakpoint () at kernel/debug/debug_core.c:1014
#1  0xc10da57b in sysrq_handle_dbg (key=<optimized out>)
    at kernel/debug/debug_core.c:773
#2  0xc13dffd5 in __handle_sysrq (key=103, check_mask=<optimized out>)
    at drivers/tty/sysrq.c:535
#3  0xc13e005b in write_sysrq_trigger (file=<optimized out>,
    buf=<optimized out>, count=2, ppos=0xf2365f98) at drivers/tty/sysrq.c:1083
#4  0xc11cb963 in proc_reg_write (file=0xf22286c0,
    buf=0x9926c08 "g\n\
```

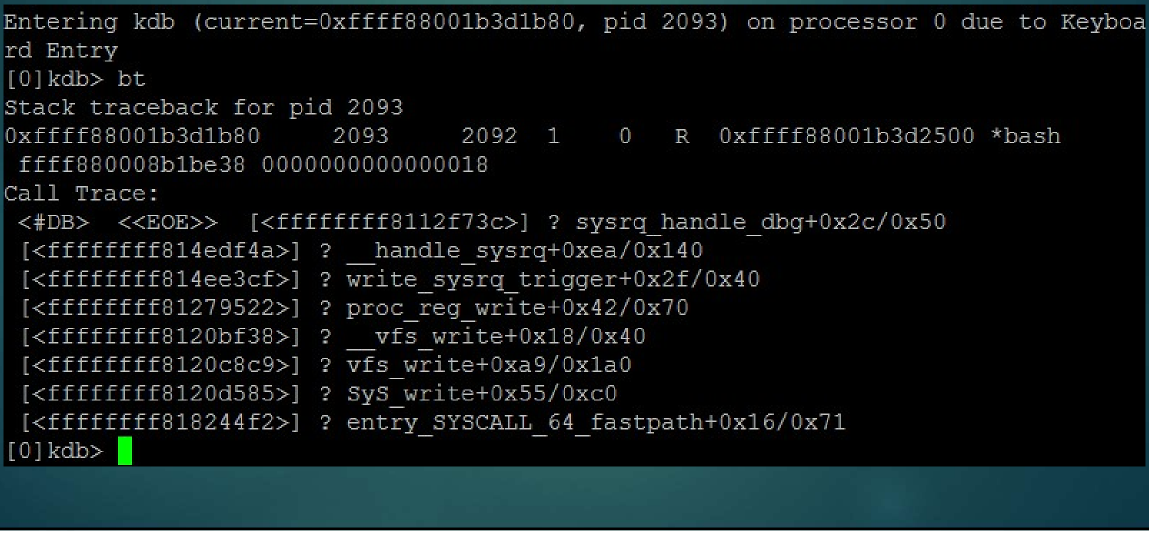
与KDB的Break-in过程非常类似



```
Entering kdb (current=0xe1c22700, pid 2822) on processor 0 due to Keyboard Entry
[0]kdb> bt
Stack traceback for pid 2822
0xe1c22700     2822     2814  1    0   R  0xe1c22a30 *bash
 e1c4ff08 c10da57b e1c4ff14 e1c4ff2c c13dffd5 c1857ad3 c18744fd 00000000
 00000004 00000002 f6a66cc0 c13e0030 e1c4ff38 c13e005b f23f0e40 e1c4ff5c
 c11cb963 e1c4ff98 e18594b0 00000002 084dec08 f23f0e40 084dec08 c11cb910
Call Trace:
 [<c10da57b>] ? sysrq_handle_dbg+0x2b/0x50
 [<c13dffd5>] ? __handle_sysrq+0xf5/0x150
 [<c13e0030>] ? __handle_sysrq+0x150/0x150
 [<c13e005b>] ? write_sysrq_trigger+0x2b/0x30
 [<c11cb963>] ? proc_reg_write+0x53/0x90
 [<c11cb910>] ? proc_reg_poll+0x80/0x80
 [<c1178645>] ? vfs_write+0xa5/0x1d0
 [<c1178af7>] ? SyS_write+0x57/0xa0
 [<c168344d>] ? sysenter_do_call+0x12/0x28
[0]kdb> 
```

```
#ifdef CONFIG_MAGIC_SYSRQ
static void sysrq_handle_dbg(int key)
{
	if (!dbg_io_ops) {
		pr_crit("ERROR: No KGDB I/O module available\n");
		return;
	}
	if (!kgdb_connected) {
#ifdef CONFIG_KGDB_KDB
		if (!dbg_kdb_mode)
			pr_crit("KGDB or $3#33 for KDB\n");
#else
		pr_crit("Entering KGDB\n");
#endif
	}

	kgdb_breakpoint();
}

static struct sysrq_key_op sysrq_dbg_op = {
	.handler  = sysrq_handle_dbg,
	.help_msg	= "debug(g)",
	.action_msg	= "DEBUG",
};
#endif
```

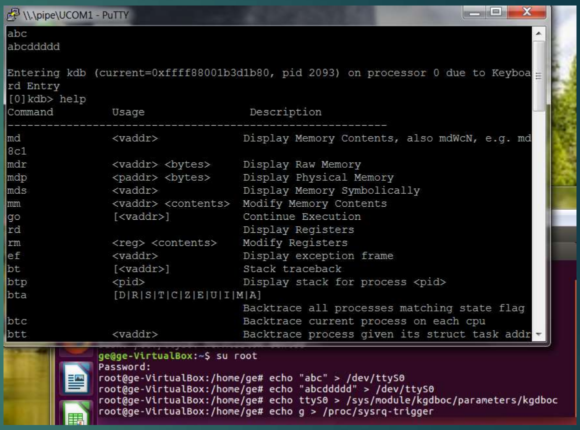
x64



```
Entering kdb (current=0xffff88001b3d1b80, pid 2093) on processor 0 due to Keyboa
rd Entry
[0]kdb> bt
Stack traceback for pid 2093
0xffff88001b3d1b80     2093     2092  1    0   R  0xffff88001b3d2500 *bash
 ffff880008b1be38 0000000000000018
Call Trace:
 <#DB>  <<EOE>>  [<ffffffff8112f73c>] ? sysrq_handle_dbg+0x2c/0x50
 [<ffffffff814edf4a>] ? __handle_sysrq+0xea/0x140
 [<ffffffff814ee3cf>] ? write_sysrq_trigger+0x2f/0x40
 [<ffffffff81279522>] ? proc_reg_write+0x42/0x70
 [<ffffffff8120bf38>] ? __vfs_write+0x18/0x40
 [<ffffffff8120c8c9>] ? vfs_write+0xa9/0x1a0
 [<ffffffff8120d585>] ? SyS_write+0x55/0xc0
 [<ffffffff818244f2>] ? entry_SYSCALL_64_fastpath+0x16/0x71
[0]kdb> 
```

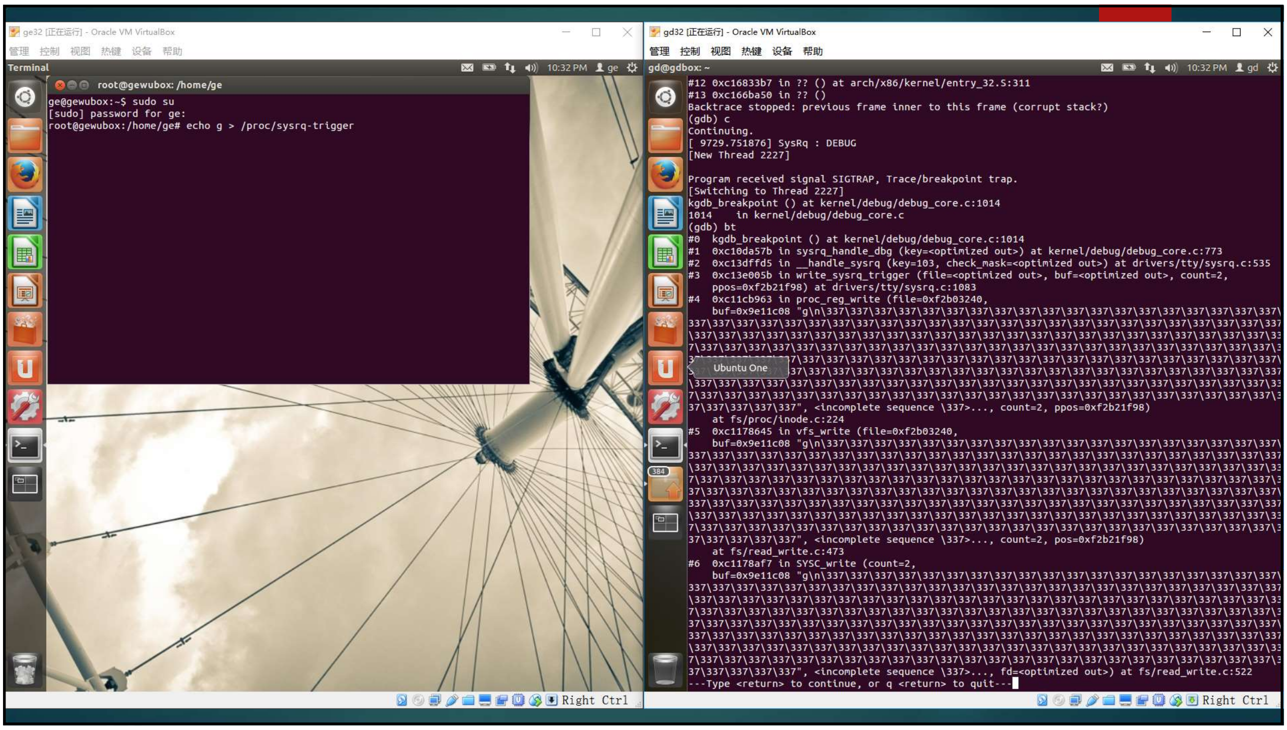
写虚拟文件

- echo g > /proc/sysrq-trigger



```
abc
abcddddd

Entering kdb (current=0xffff88001b3d1b80, pid 2093) on processor 0 due to Keyboa
rd Entry
[0]kdb> help
Command         Usage                Description
----------------------------------------------------------
md              <vaddr>             Display Memory Contents, also mdWcN, e.g. md
8c1
mdr             <vaddr> <bytes>     Display Raw Memory
mdp             <paddr> <bytes>     Display Physical Memory
mds             <vaddr>             Display Memory Symbolically
mm              <vaddr> <contents>  Modify Memory Contents
go              [<vaddr>]           Continue Execution
rd                                  Display Registers
rm              <reg> <contents>    Modify Registers
ef              <vaddr>             Display exception frame
bt              [<vaddr>]           Stack traceback
btp             <pid>               Display stack for process <pid>
bta             [D|R|S|T|C|Z|E|U|I|M|A]
                                    Backtrace all processes matching state flag
btc                                 Backtrace current process on each cpu
btt             <vaddr>             Backtrace process given its struct task addr
```

```
ge@ge-VirtualBox:~$ su root
Password:
root@ge-VirtualBox:/home/ge# echo "abc" > /dev/ttyS0
root@ge-VirtualBox:/home/ge# echo "abcddddd" > /dev/ttyS0
root@ge-VirtualBox:/home/ge# echo ttyS0 > /sys/module/kgdboc/parameters/kgdboc
root@ge-VirtualBox:/home/ge# echo g > /proc/sysrq-trigger
```


ge32 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Terminal
10:32 PM
ge
root@gewubox: /home/ge
Right Ctrl
gd32 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
gd@gdbox: ~
10:32 PM
gd
Ubuntu One
Right Ctrl

```
ge@gewubox:~$ sudo su
[sudo] password for ge:
root@gewubox:/home/ge# echo g > /proc/sysrq-trigger
```

```
#12 0xc16833b7 in ?? () at arch/x86/kernel/entry_32.S:311
#13 0xc166ba50 in ?? ()
Backtrace stopped: previous frame inner to this frame (corrupt stack?)
(gdb) c
Continuing.
[ 9729.751876] SysRq : DEBUG
[New Thread 2227]

Program received signal SIGTRAP, Trace/breakpoint trap.
[Switching to Thread 2227]
kgdb_breakpoint () at kernel/debug/debug_core.c:1014
1014    in kernel/debug/debug_core.c
(gdb) bt
#0  kgdb_breakpoint () at kernel/debug/debug_core.c:1014
#1  0xc10da57b in sysrq_handle_dbg (key=<optimized out>) at kernel/debug/debug_core.c:773
#2  0xc13dffd5 in __handle_sysrq (key=103, check_mask=<optimized out>) at drivers/tty/sysrq.c:535
#3  0xc13e005b in write_sysrq_trigger (file=<optimized out>, buf=<optimized out>, count=2,
    ppos=0xf2b21f98) at drivers/tty/sysrq.c:1083
#4  0xc11cb963 in proc_reg_write (file=0xf2b03240,
    buf=0x9e11c08 "g\n\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\
337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337
\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\33
7\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\3
37\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\
337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337
\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\33
7\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\3
37\337\337\337\337", <incomplete sequence \337>..., count=2, ppos=0xf2b21f98)
    at fs/proc/inode.c:224
#5  0xc1178645 in vfs_write (file=0xf2b03240,
    buf=0x9e11c08 "g\n\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\
337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337
\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\33
7\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\3
37\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\
337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337
\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\33
7\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\3
37\337\337\337\337", <incomplete sequence \337>..., count=2, pos=0xf2b21f98)
    at fs/read_write.c:473
#6  0xc1178af7 in SYSC_write (count=2,
    buf=0x9e11c08 "g\n\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\
337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337
\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\33
7\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\3
37\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\
337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337
\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\33
7\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\337\3
37\337\337\337\337", <incomplete sequence \337>..., fd=<optimized out>) at fs/read_write.c:522
---Type <return> to continue, or q <return> to quit---
```

工作过程

```
breakpoint causes exception
  exception handler saves state
  exception handler calls gdb stub
    kgdb_handle_exception()
      if (kgdb_active) return 0
      marshall other processors
      put packet to host -- exception occurred
      do
        get command packet from host (polled wait)
        process command in packet
        put reply packet to host
      while (command != 'c', 's', 'D', 'k', 'r')
      # continue, single step, detach, kill, reset
      release other processors
  restore state, return to execute instruction that caused break
```

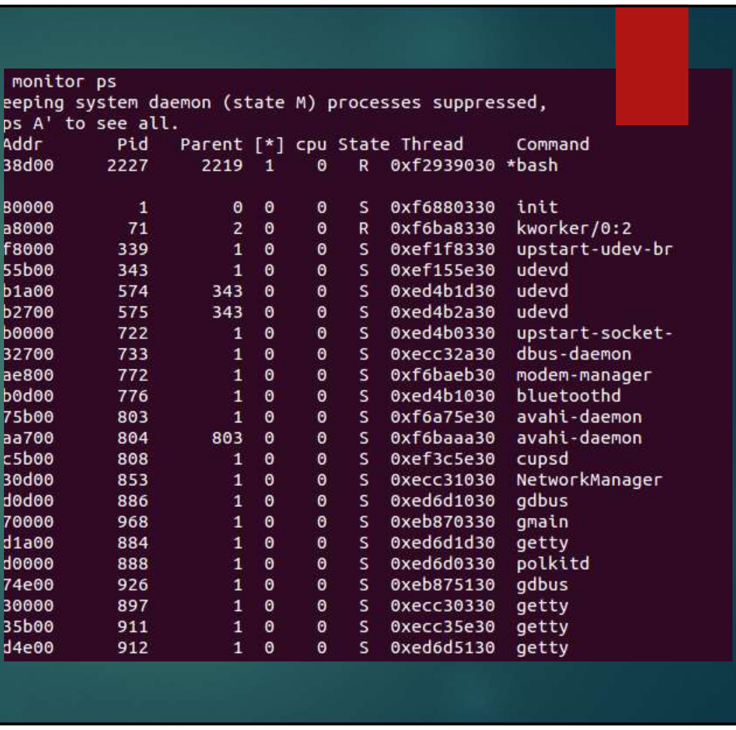
* Jason Wessel

适用命令

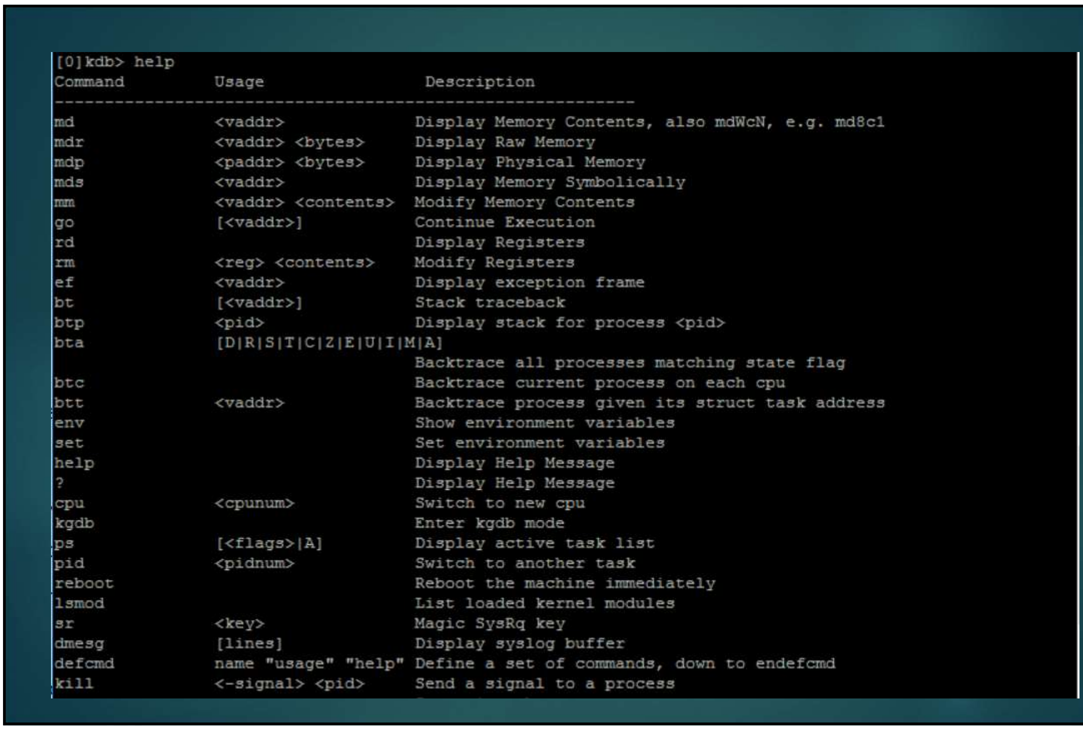
- 大多数GDB命令都可以与普通GDB一样使用
  - 断点
  - bt
  - 观察变量
  - 观察内存
- 设置符号时有所不同
  - 内核加载在固定地址，可以直接指定
  - 可加载模块需要指定内存布局参数
- 使用KGDB时，把整个系统大致看作一个进程

使用kdb命令

- 通过monitor命令使用一部分kdb命令

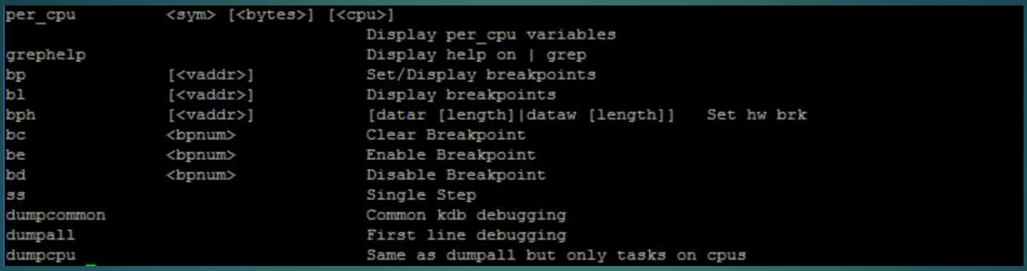


```
 monitor ps
eeping system daemon (state M) processes suppressed,
ps A' to see all.
Addr        Pid   Parent [*] cpu State Thread     Command
38d00      2227     2219  1    0   R  0xf2939030 *bash

80000         1        0  0    0   S  0xf6880330  init
a8000        71        2  0    0   R  0xf6ba8330  kworker/0:2
f8000       339        1  0    0   S  0xef1f8330  upstart-udev-br
55b00       343        1  0    0   S  0xef155e30  udevd
b1a00       574      343  0    0   S  0xed4b1d30  udevd
b2700       575      343  0    0   S  0xed4b2a30  udevd
b0000       722        1  0    0   S  0xed4b0330  upstart-socket-
32700       733        1  0    0   S  0xecc32a30  dbus-daemon
ae800       772        1  0    0   S  0xf6baeb30  modem-manager
b0d00       776        1  0    0   S  0xed4b1030  bluetoothd
75b00       803        1  0    0   S  0xf6a75e30  avahi-daemon
aa700       804      803  0    0   S  0xf6baaa30  avahi-daemon
c5b00       808        1  0    0   S  0xef3c5e30  cupsd
30d00       853        1  0    0   S  0xecc31030  NetworkManager
d0d00       886        1  0    0   S  0xed6d1030  gdbus
70000       968        1  0    0   S  0xeb870330  gmain
d1a00       884        1  0    0   S  0xed6d1d30  getty
d0000       888        1  0    0   S  0xed6d0330  polkitd
74e00       926        1  0    0   S  0xeb875130  gdbus
30000       897        1  0    0   S  0xecc30330  getty
35b00       911        1  0    0   S  0xecc35e30  getty
d4e00       912        1  0    0   S  0xed6d5130  getty
```



```
[0]kdb> help
Command         Usage                Description
----------------------------------------------------------
md              <vaddr>             Display Memory Contents, also mdWcN, e.g. md8c1
mdr             <vaddr> <bytes>     Display Raw Memory
mdp             <paddr> <bytes>     Display Physical Memory
mds             <vaddr>             Display Memory Symbolically
mm              <vaddr> <contents>  Modify Memory Contents
go              [<vaddr>]           Continue Execution
rd                                  Display Registers
rm              <reg> <contents>    Modify Registers
ef              <vaddr>             Display exception frame
bt              [<vaddr>]           Stack traceback
btp             <pid>               Display stack for process <pid>
bta             [D|R|S|T|C|Z|E|U|I|M|A]
                                    Backtrace all processes matching state flag
btc                                 Backtrace current process on each cpu
btt             <vaddr>             Backtrace process given its struct task address
env                                 Show environment variables
set                                 Set environment variables
help                                Display Help Message
?                                   Display Help Message
cpu             <cpunum>            Switch to new cpu
kgdb                                Enter kgdb mode
ps              [<flags>|A]         Display active task list
pid             <pidnum>            Switch to another task
reboot                              Reboot the machine immediately
lsmod                               List loaded kernel modules
sr              <key>               Magic SysRq key
dmesg           [lines]             Display syslog buffer
defcmd          name "usage" "help" Define a set of commands, down to endefcmd
kill            <-signal> <pid>     Send a signal to a process
```

续

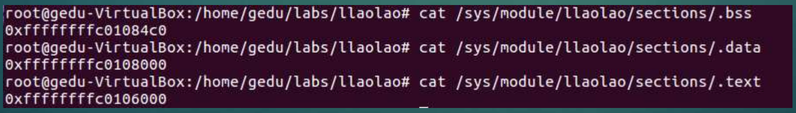


```
per_cpu         <sym> [<bytes>] [<cpu>]
                                    Display per_cpu variables
grephelp                            Display help on | grep
bp              [<vaddr>]           Set/Display breakpoints
bl              [<vaddr>]           Display breakpoints
bph             [<vaddr>]           [datar [length]|dataw [length]]   Set hw brk
bc              <bpnum>             Clear Breakpoint
be              <bpnum>             Enable Breakpoint
bd              <bpnum>             Disable Breakpoint
ss                                  Single Step
dumpcommon                          Common kdb debugging
dumpall                             First line debugging
dumpcpu                             Same as dumpall but only tasks on cpus
```

- B系列和WinDBG非常类似

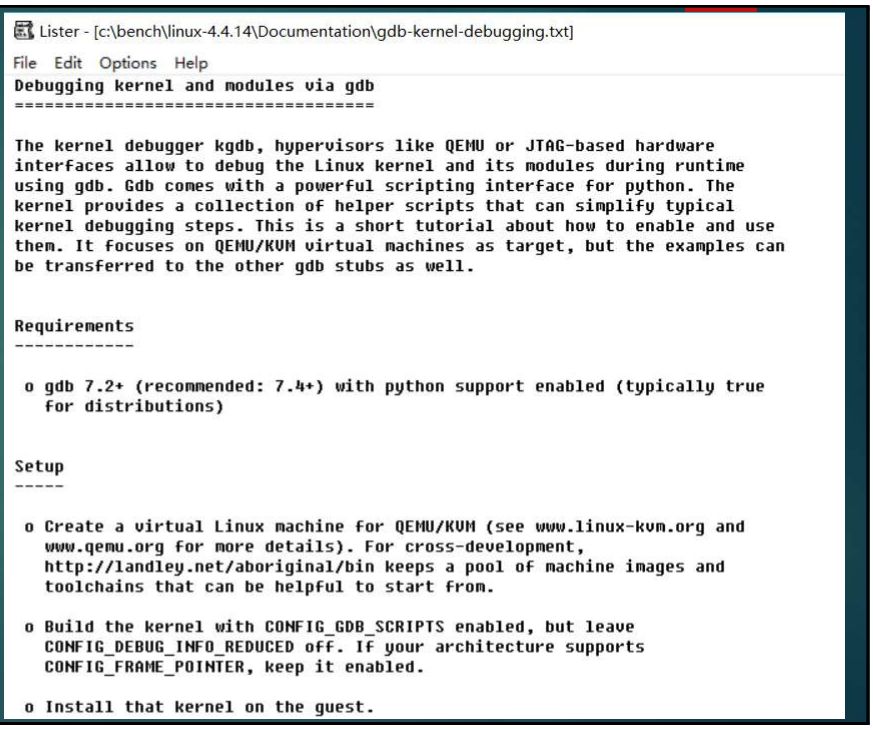
add-symbol-file

(gdb) add-symbol-file /media/sf_temp/ge/e1000.ko 0xf858c000 -s .bss 0xf85a6ba4 -s .data 0xf85a5000
add symbol table from file "/media/sf_temp/ge/e1000.ko" at
    .text_addr = 0xf858c000
    .bss_addr = 0xf85a6ba4
    .data_addr = 0xf85a5000
(y or n) y
Reading symbols from /media/sf_temp/ge/e1000.ko...done.



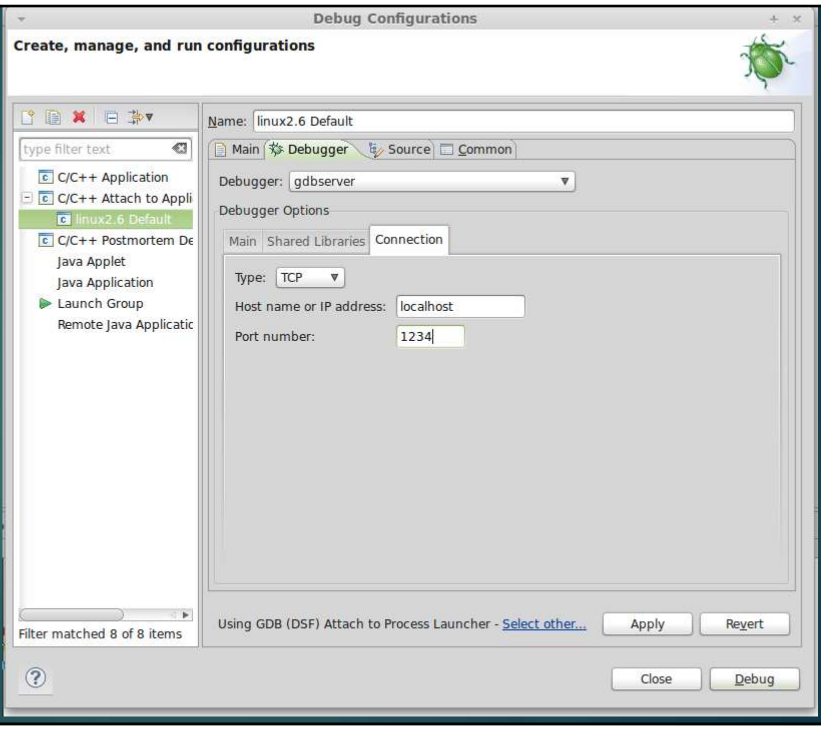
```
root@gedu-VirtualBox:/home/gedu/labs/llaolao# cat /sys/module/llaolao/sections/.bss
0xffffffffc01084c0
root@gedu-VirtualBox:/home/gedu/labs/llaolao# cat /sys/module/llaolao/sections/.data
0xffffffffc0108000
root@gedu-VirtualBox:/home/gedu/labs/llaolao# cat /sys/module/llaolao/sections/.text
0xffffffffc0106000
```

Python脚本


Lister - [c:\bench\linux-4.4.14\Documentation\gdb-kernel-debugging.txt]
File
Edit
Options
Help

```
Debugging kernel and modules via gdb
====================================

The kernel debugger kgdb, hypervisors like QEMU or JTAG-based hardware
interfaces allow to debug the Linux kernel and its modules during runtime
using gdb. Gdb comes with a powerful scripting interface for python. The
kernel provides a collection of helper scripts that can simplify typical
kernel debugging steps. This is a short tutorial about how to enable and use
them. It focuses on QEMU/KVM virtual machines as target, but the examples can
be transferred to the other gdb stubs as well.


Requirements
------------

 o gdb 7.2+ (recommended: 7.4+) with python support enabled (typically true
   for distributions)


Setup
-----

 o Create a virtual Linux machine for QEMU/KVM (see www.linux-kvm.org and
   www.qemu.org for more details). For cross-development,
   http://landley.net/aboriginal/bin keeps a pool of machine images and
   toolchains that can be helpful to start from.

 o Build the kernel with CONFIG_GDB_SCRIPTS enabled, but leave
   CONFIG_DEBUG_INFO_REDUCED off. If your architecture supports
   CONFIG_FRAME_POINTER, keep it enabled.

 o Install that kernel on the guest.
```

图形前端



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Appli
  - linux2.6 Default
- C/C++ Postmortem De
- Java Applet
- Java Application
- Launch Group
- Remote Java Applicatic

Filter matched 8 of 8 items

Name: linux2.6 Default

Main | Debugger | Source | Common

Debugger: gdbserver

Debugger Options

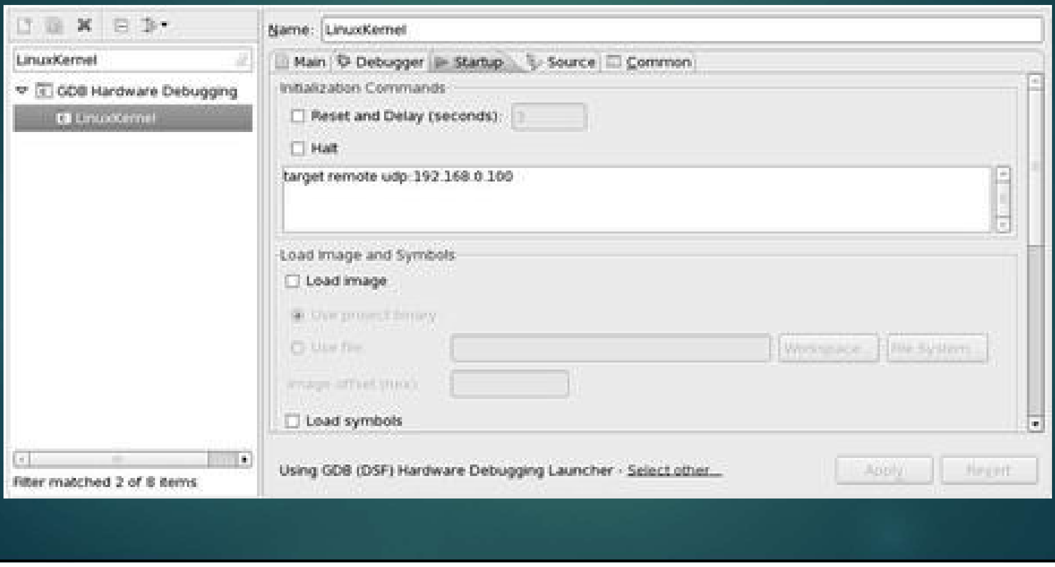
Main | Shared Libraries | Connection

Type: TCP

Host name or IP address: localhost

Port number: 1234

Using GDB (DSF) Attach to Process Launcher - Select other...



LinuxKernel

- GDB Hardware Debugging
  - LinuxKernel

Filter matched 2 of 8 items

Name: LinuxKernel

Main | Debugger | Startup | Source | Common

Initialization Commands

□ Reset and Delay (seconds):

□ Halt

```
target remote udp:192.168.0.100
```

Load Image and Symbols

□ Load image

● Use project binary

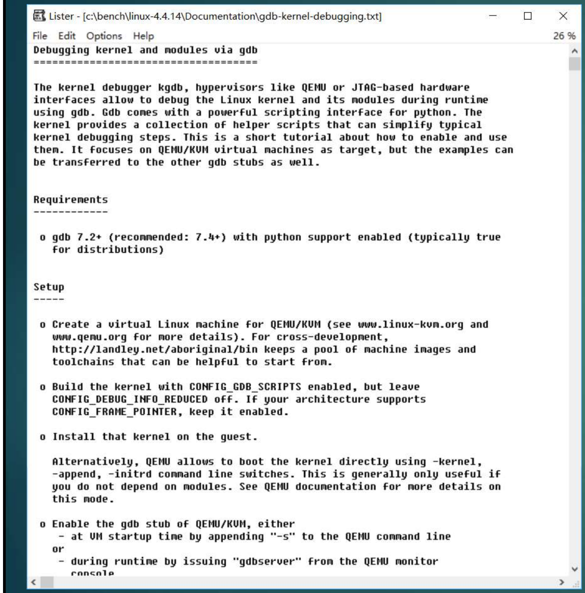
○ Use file

Image offset (hex):

□ Load symbols

Using GDB (DSF) Hardware Debugging Launcher - Select other...

KGDB via QEMU


Lister - [c:\bench\linux-4.4.14\Documentation\gdb-kernel-debugging.txt]
File Edit Options Help
26 %

```
Debugging kernel and modules via gdb
====================================

The kernel debugger kgdb, hypervisors like QEMU or JTAG-based hardware
interfaces allow to debug the Linux kernel and its modules during runtime
using gdb. Gdb comes with a powerful scripting interface for python. The
kernel provides a collection of helper scripts that can simplify typical
kernel debugging steps. This is a short tutorial about how to enable and use
them. It focuses on QEMU/KVM virtual machines as target, but the examples can
be transferred to the other gdb stubs as well.


Requirements
------------

 o gdb 7.2+ (recommended: 7.4+) with python support enabled (typically true
   for distributions)


Setup
-----

 o Create a virtual Linux machine for QEMU/KVM (see www.linux-kvm.org and
   www.qemu.org for more details). For cross-development,
   http://landley.net/aboriginal/bin keeps a pool of machine images and
   toolchains that can be helpful to start from.

 o Build the kernel with CONFIG_GDB_SCRIPTS enabled, but leave
   CONFIG_DEBUG_INFO_REDUCED off. If your architecture supports
   CONFIG_FRAME_POINTER, keep it enabled.

 o Install that kernel on the guest.

   Alternatively, QEMU allows to boot the kernel directly using -kernel,
   -append, -initrd command line switches. This is generally only useful if
   you do not depend on modules. See QEMU documentation for more details on
   this mode.

 o Enable the gdb stub of QEMU/KVM, either
    - at VM startup time by appending "-s" to the QEMU command line
   or
    - during runtime by issuing "gdbserver" from the QEMU monitor
      console
```

```
sudo ./qemu-1.7.0/arm-softmmu/qemu-system-arm
-M vexpress-a9 -kernel ./linux-
3.10.28/arch/arm/boot/zImage
 -sd rootfs.img --append "root=/dev/mmcblk0 rw
rootfs=ext3 rootdelay=3  physmap.enabled=0
console=ttyAMA0
console=tty0 kgdboc=tty0,115200 kgdbwait"  -net
nic,vlan=0 -net tap,vlan=0  -serial tcp::4321,server
```

```
(gdb) target remote localhost:4321
```

59

概览

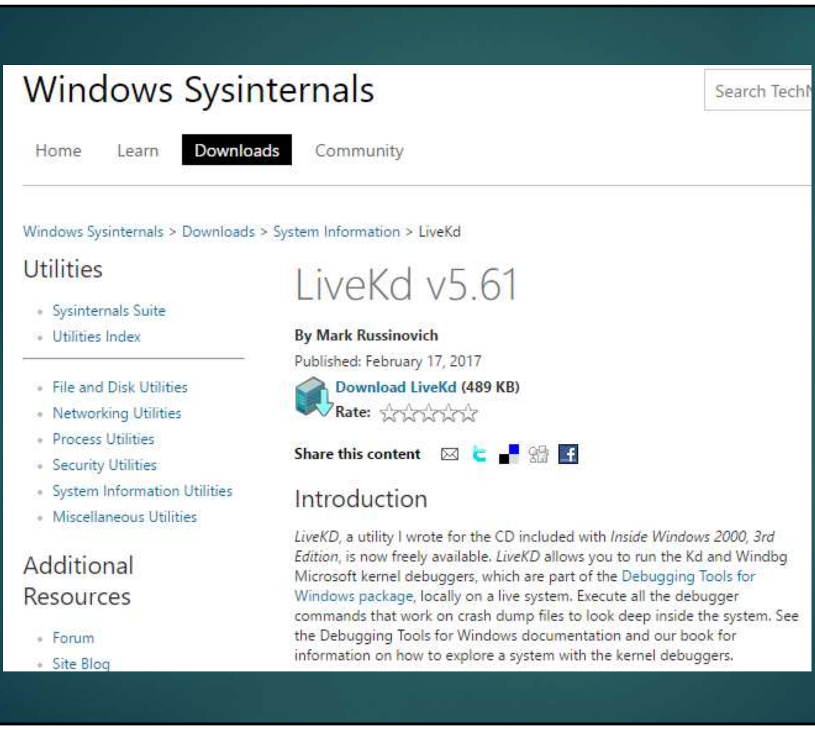
准备符号文件

启用和连接

双机调试

kcore

60



Windows Sysinternals

Search TechN

Windows Sysinternals > Downloads > System Information > LiveKd

Utilities

- Sysinternals Suite
- Utilities Index

- File and Disk Utilities
- Networking Utilities
- Process Utilities
- Security Utilities
- System Information Utilities
- Miscellaneous Utilities

Additional Resources

- Forum
- Site Blog

LiveKd v5.61

By Mark Russinovich

Published: February 17, 2017

Download LiveKd (489 KB)

Rate: ☆☆☆☆☆

Share this content

Introduction

LiveKD, a utility I wrote for the CD included with *Inside Windows 2000, 3rd Edition*, is now freely available. *LiveKD* allows you to run the Kd and Windbg Microsoft kernel debuggers, which are part of the Debugging Tools for Windows package, locally on a live system. Execute all the debugger commands that work on crash dump files to look deep inside the system. See the Debugging Tools for Windows documentation and our book for information on how to explore a system with the kernel debuggers.

Execute all the debugger commands that work on crash dump files to look deep inside the system.

使用/proc/kcore单机调试

```
ge@gewubox:~$ sudo gdb --core=/proc/kcore
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/>.
[New process 1]
Core was generated by `BOOT_IMAGE=/boot/vmlinuz-3.12.2 root=UUID=eb5d8aee-6a0f-4257-
b5b8-8d6731ba6764'.
#0  0x00000000 in ?? ()
```

- 与Windows下的Live KD类似

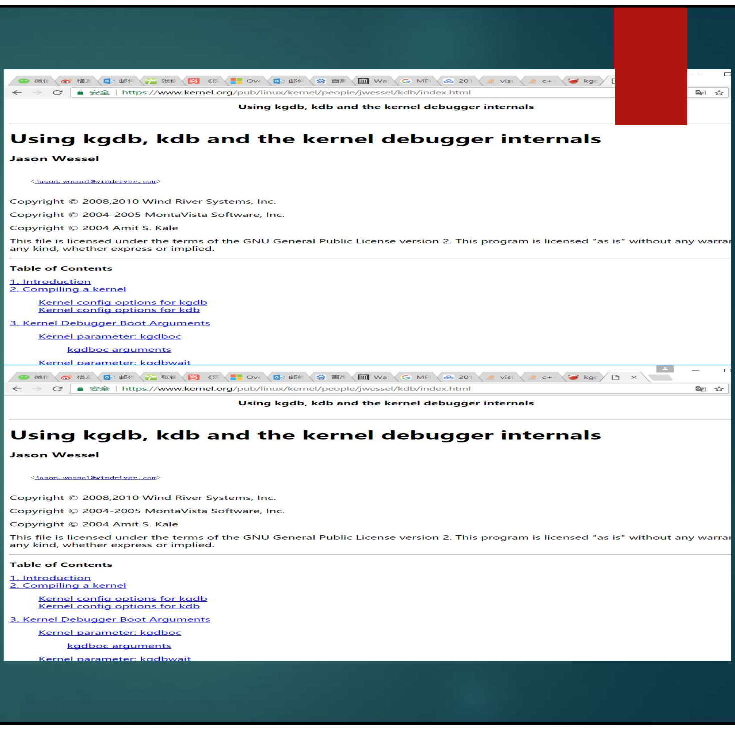
https://technet.microsoft.com/en-us/sysinternals/livekd.aspx

观察IDT表

```
(gdb) print /x idt_table[0]
$4 = {{{a = 0x60c7e4, b = 0xc1678e00}, {limit0 = 0xc7e4, base0 = 0x60,
      base1 = 0x0, type = 0xe, s = 0x0, dpl = 0x0, p = 0x1, limit = 0x7,
      avl = 0x0, l = 0x1, d = 0x1, g = 0x0, base2 = 0xc1}}}
(gdb) i symbol 0xc167c7e4
divide_error in section .text
```

文档

- https://www.kernel.org/pub/linux/kernel/people/jwessel/kdb/index.html


微信
格式
邮件
张钰
《济
Ove
邮件
百度
Wa
MF
201
visu
c++
kgd
安全
https://www.kernel.org/pub/linux/kernel/people/jwessel/kdb/index.html

Using kgdb, kdb and the kernel debugger internals

Using kgdb, kdb and the kernel debugger internals

Jason Wessel

<jason.wessel@windriver.com>

Copyright © 2008,2010 Wind River Systems, Inc.

Copyright © 2004-2005 MontaVista Software, Inc.

Copyright © 2004 Amit S. Kale

This file is licensed under the terms of the GNU General Public License version 2. This program is licensed "as is" without any warra any kind, whether express or implied.

Table of Contents

Using kgdb, kdb and the kernel debugger internals

Using kgdb, kdb and the kernel debugger internals

Jason Wessel

<jason.wessel@windriver.com>

Copyright © 2008,2010 Wind River Systems, Inc.

Copyright © 2004-2005 MontaVista Software, Inc.

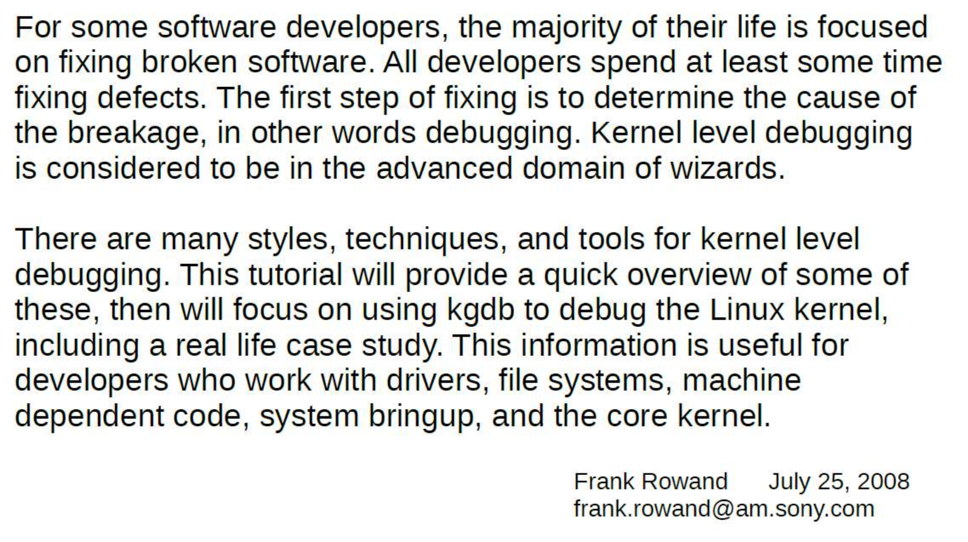
Copyright © 2004 Amit S. Kale

This file is licensed under the terms of the GNU General Public License version 2. This program is licensed "as is" without any warra any kind, whether express or implied.

Table of Contents

1. Introduction
2. Compiling a kernel
Kernel config options for kgdb
Kernel config options for kdb
3. Kernel Debugger Boot Arguments
Kernel parameter: kgdboc
kgdboc arguments
Kernel parameter: kgdbwait

Pitfalls of a Kernel Debugger



For some software developers, the majority of their life is focused on fixing broken software. All developers spend at least some time fixing defects. The first step of fixing is to determine the cause of the breakage, in other words debugging. Kernel level debugging is considered to be in the advanced domain of wizards.

There are many styles, techniques, and tools for kernel level debugging. This tutorial will provide a quick overview of some of these, then will focus on using kgdb to debug the Linux kernel, including a real life case study. This information is useful for developers who work with drivers, file systems, machine dependent code, system bringup, and the core kernel.

Frank Rowand July 25, 2008
frank.rowand@am.sony.com

Frank Rowand

- Sony experts present new kernel research and a tool to measure wake up latency at LinuxCon Japan
- Next week, experienced Linux developers **Frank Rowand** (left) and **Tim Bird** (right) from Sony will be presenting atLinuxCon Japan, where many top Linux developers, administrators, users and experts will come together. Frank will share insights on how to use and understand the Real-Time Cyclictest benchmark, and Tim will explain how Linux developers can reduce the size of their Linux systems (and possible improve their performance in the process). Learn more the event and the presentations after the jump!

```
(gdb) disassemble SYSC_reboot
Dump of assembler code for function SYSC_reboot:
   0xffffffff810a6750 <+0>:push   %rbp
   0xffffffff810a6751 <+1>:mov    %rsp,%rbp
   0xffffffff810a6754 <+4>:push   %r15
   0xffffffff810a6756 <+6>:push   %r14
   0xffffffff810a6758 <+8>:push   %r13
[...]
  0xffffffff810a67a2 <+82>:     je     0xffffffff810a6957 <SYSC_reboot+519>
   0xffffffff810a67a8 <+88>:    cmp    $0xfee1dead,%r15d
   0xffffffff810a67af <+95>:    jne    0xffffffff810a68fa <SYSC_reboot+426>
   0xffffffff810a67b5 <+101>:   cmp    $0x28121969,%ebx
---Type <return> to continue, or q <return> to quit---
   0xffffffff810a67bb <+107>:   je     0xffffffff810a67d9 <SYSC_reboot+137>
   0xffffffff810a67bd <+109>:   cmp    $0x5121996,%ebx
   0xffffffff810a67c3 <+115>:   je     0xffffffff810a67d9 <SYSC_reboot+137>
   0xffffffff810a67c5 <+117>:   cmp    $0x16041998,%ebx
   0xffffffff810a67cb <+123>:   je     0xffffffff810a67d9 <SYSC_reboot+137>
   0xffffffff810a67cd <+125>:   cmp    $0x20112000,%ebx
   0xffffffff810a67d3 <+131>:   jne    0xffffffff810a68fa <SYSC_reboot+426>
   0xffffffff810a67d9 <+137>:   mov    %r12d,%esi
   0xffffffff810a67dc <+140>:   mov    %r13,%rdi
```

切问而近思

欢迎关注格友公众号