Foswiki > Main Web > DebuggingTechniques > KernelOOPS (11 Oct 2014, AdminUser)

Kernel Masters

http://www.kernelmasters.org

How to Debug Kernel Module using kgdb

Example1: Character Device Driver Template

Target Side:

Step1: Compile Kernel module with debug symbols

Add KBUILD_CFLAGS += -g flag in top most Makefile in Kernel source code.

Character Device Driver Template source code

* char.c: char.c

* Makefile: Makefile

$ make

Step2: Copy vmlinux, Module source code and .ko files in to host

$ scp vmlinux username@IPAddress:/home//kgdb_kdb_images

$ scp -r "" username@IPAddress:/home//kgdb_kdb_images

$ sudo insmod char.ko

$ sudo mknod /dev/mychar c 60 0

$ sudo chmod 777 /dev/mychar

Step3: Identify Section addresses

$ sudo su // Should be in root user

cd /sys/modules/char/sections

ls -la

cat .text .data .bss

Step4: Setup KGDB breakpoint

$ sudo su

echo g > /proc/sysrq-trigger

Host Side: Load kernel module using gdb

$ cd ~ // Enter home directory

$ mkdir kgdb_kdb_images

kernelmasters@km-desktop:~$ gdb kgdb_kdb_images/vmlinux

GNU gdb (Ubuntu 7.7-0ubuntu3.1) 7.7 Copyright (C) 2014 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". Type "show configuration" for configuration details. For bug reporting instructions, please see: <http://www.gnu.org/software/gdb/bugs/>. Find the GDB manual and other documentation resources online at: <http://www.gnu.org/software /gdb/documentation/>. For help, type "help". Type "apropos word" to search for commands related to "word"... Reading symbols from kgdb_kdb_images/vmlinux...done.

(gdb) target remote /dev/pts/11

Remote debugging using /dev/pts/11

kgdb_breakpoint () at kernel/debug/debug_core.c:1043

1043 kernel/debug/debug_core.c: No such file or directory.

(gdb) add-symbol-file kgdb_kdb_images/2_CharTemp/char.ko 0xffffffffa00e4000 -s .data 0xffffffffa00e6000 -s .bss 0xffffffffa00e6360 -s .rodata.str1.1 0xffffffffa00e5024 -s .rodata.str1.8 0xffffffffa00e5088

add symbol table from file "kgdb_kdb_images/2_CharTemp/char.ko" at

.text_addr = 0xffffffffa00e4000

.data_addr = 0xffffffffa00e6000

.bss_addr = 0xffffffffa00e6360

.rodata.str1.1_addr = 0xffffffffa00e5024

.rodata.str1.8_addr = 0xffffffffa00e5088

(y or n) y

Reading symbols from kgdb_kdb_images/2_CharTemp/char.ko...done.

(gdb) b my_open (or) hb my_open

Breakpoint 1 at 0xffffffffa00e4050: file /home/km/4_CDD/2_CharTemp/char.c, line 56.

hb -- Hardware breakpoint

Set a hardware-assisted breakpoint. The args are the same as for the break command and the breakpoint is set in the same way, but the breakpoint requires hardware support and some target hardware may not have this support

(gdb) b my_read

Breakpoint 2 at 0xffffffffa00e4000: file /home/km/4_CDD/2_CharTemp/char.c, line 43.

(gdb) b my_write

Breakpoint 3 at 0xffffffffa00e4020: file /home/km/4_CDD/2_CharTemp/char.c, line 50.

(gdb) c

Continuing.

For More information see the below log file:

- module_debugging.log2: debugging.log

KDB to KGDB

Enter kgdb in kdb command prompt than close serial port application.

kdb> kgdb

KGDB to KDB

Open serial port and Enter $3#33 in serial port. And close kgdb application.

Example2: Panic Example

gdb oops.o

.....

Reading symbols from /home/kernelmasters/KM_GIT/debugging/kdb_kgdb/kernel_panic/panic.o...done.

(gdb) l #include <linux/kernel.h>

#include <linux/module.h>

#include <linux/init.h>

static void create_oops() {

*(int *)0 = 0;

}

static int __init my_oops_init(void) {

printk("oops from the module\n");

create_oops();

return (0);

}

static void __exit my_oops_exit(void) {

printk("Goodbye world\n");

}

module_init(my_oops_init);

module_exit(my_oops_exit);

(gdb) disassemble my_oops_init Dump of assembler code for function my_oops_init:

1. x0000000000000025 <+0>: push %rbp
2. x0000000000000026 <+1>: mov $0x0,%rdi
3. x000000000000002d <+8>: xor %eax,%eax
4. x000000000000002f <+10>: mov %rsp,%rbp
5. x0000000000000032 <+13>: callq 0x37 <my_oops_init+18>
6. x0000000000000037 <+18>: movl $0x0,0x0
7. x0000000000000042 <+29>: xor %eax,%eax
8. x0000000000000044 <+31>: pop %rbp

9. x0000000000000045 <+32>: retq

End of assembler dump.

(gdb) list *(0x0000000000000037)

0x37 is in my_oops_init (/home/kernelmasters/KM_GIT/debugging/kdb_kgdb/kernel_panic/panic.c:6).

1 #include <linux/kernel.h>

2 #include <linux/module.h>

3 #include <linux/init.h>

4

5 static void create_oops() {

6 *(int *)0 = 0;

7 }

8 static int __init my_oops_init(void) {

9 printk("oops from the module\n");

* kernel_panic_debug.log: kernel Panic Debug log

$ sudo insmod panic.ko

- kernel_panic.log: kernel_panic

Assignment - Intel Interview Question

Question

Please locate the exact line in the sourcecode which caused the panic. Crash.txt has a Kernel crash message. hv.zip has the source code.

- hv.zip: Crase_Source.zip
- crash.txt: crash.txt

http://www.kernelmasters.org

-- KishoreBoddu - 07 Sep 2013

- panic_char.log: panic_char.log

Edit | Attach | Print version | History: r7 < r6 < r5 < r4 | Backlinks | View wiki text | Edit wiki text | More topic actions

Topic revision: r7 - 11 Oct 2014, AdminUser