Linux Kernel Debugging

Dongdong Deng <LibFetion@gmail.com>

KGDB.info

Overview of Talks

- Kernel Problems
- Collect System Info
- Handling Failures
- Debugging Techniques
- Crash Analyse
- Debugging Process
- Debugging Tricks

Kernel Problems

- Root cause of problems
  - self problem (Logic Implementation)
  - cooperating problem (incorrect API/Function usage)
  - platform problem (hardware)

- Phenomenon
  - system behave incorrectly
  - oops/panic
  - system hang

Collect System Info

- System error logs
  - dmesg #dmesg | tail
  - /var/log/ #ls /var/log/
- Console
  - local console
  - remote console
- Others
  - log by programer

KGDB.info

Handling Failures

- System behave incorrectly
  - compare with normal behavior
  - analyze and fix
- System Crash
  - collect and analyze oops/painc data
  - collect and analyze dump data
- System Hang
  - look at the hang using ICE/JTAG
  - trigger magic sysreq keys
  - look at the hang using kgdb/kdb(If possible)
  - hacking codes to use NMI features (if support)

Debugging Techniques

- Basic
  - Printk()
- **Best**
  - JTAG, ICE,
- Better
  - Virtual Machine backend debugger
  - Kdump/Kexec
- Good
  - KGDB / KDB
- Others
  - Kprobe
  - Perf
  - Ftrace.. so on..

printk()

- Works like *printf()*
  - printk(KERN_DEBUG ”Get printk: %s:%i\n”, __FILE__, __LINE__);
  - printk(KERN_CRIT "OOO at %p\n", pointer);

- Output with priorities
  - KERN_ERR, KERN_WARNING, KERN_INFO, so on…
  - pr_err().pr_warning(),pr_info()…

```
#define pr_err(fmt, …) \
printk(KERN_ERR pr_fmt(fmt), ##__VA_ARGS_)
```

- Increase Log buffer
  - CONFIG_LOG_BUF_SHIFT

- Modify the console printk level
  - #echo 8 > /proc/sys/kernel/printk or #dmesg -n 8
  - integers range from 0 to 7, with smaller values representing higher priorities.

KGDB.info

How printk() work

printk() can be called from any context. **Why?**

```
void printk() {
spin_lock(&logbuf_lock);

emit_log_char() --> add data to logbuf

if (!down_trylock(&console_sem)) {
    spin_unlock(&logbuf_lock);
    return;
}

spin_unlock(&logbuf_lock);

release_console_sem();
}
```

console --> output device
logbuf --> a store buffer for printk data
logbuf_lock -> an spinlock for operating logbuf
console_sem -> an semaphore for operating console device

How printk() work

```
void release_console_sem() {

for (; ;) {
spin_lock(&logbuf_lock);
if (logbuf_start == logbuf_end)
        break;

out_start = logbuf_start;    out_end = logbuf_end;
spin_unlock(&logbuf_lock);

call_console_device (out_start, out_end);
}

up(&console_sem);
spin_unlock (&logbuf_lock);
}
```

How printk() work

```
printk() {

spin_lock(&logbuf_lock);
emit_log_char(logbuf);
spin_unlock(&logbuf_lock);

down (&console_sem));

spin_lock(&logbuf_lock);
call_console_device (logbuf);  → write output device…
spin_unlock(&logbuf_lock);

up(&console_sem);
}
```

printk()

- **advantages**
  - easy using
  - not need any other system support
- **disadvantages**
  - have to modify/rebuild source
  - cann't debug online Interactively
  - affect time / behavior
  - working linear
- **Do we need a debugger?**

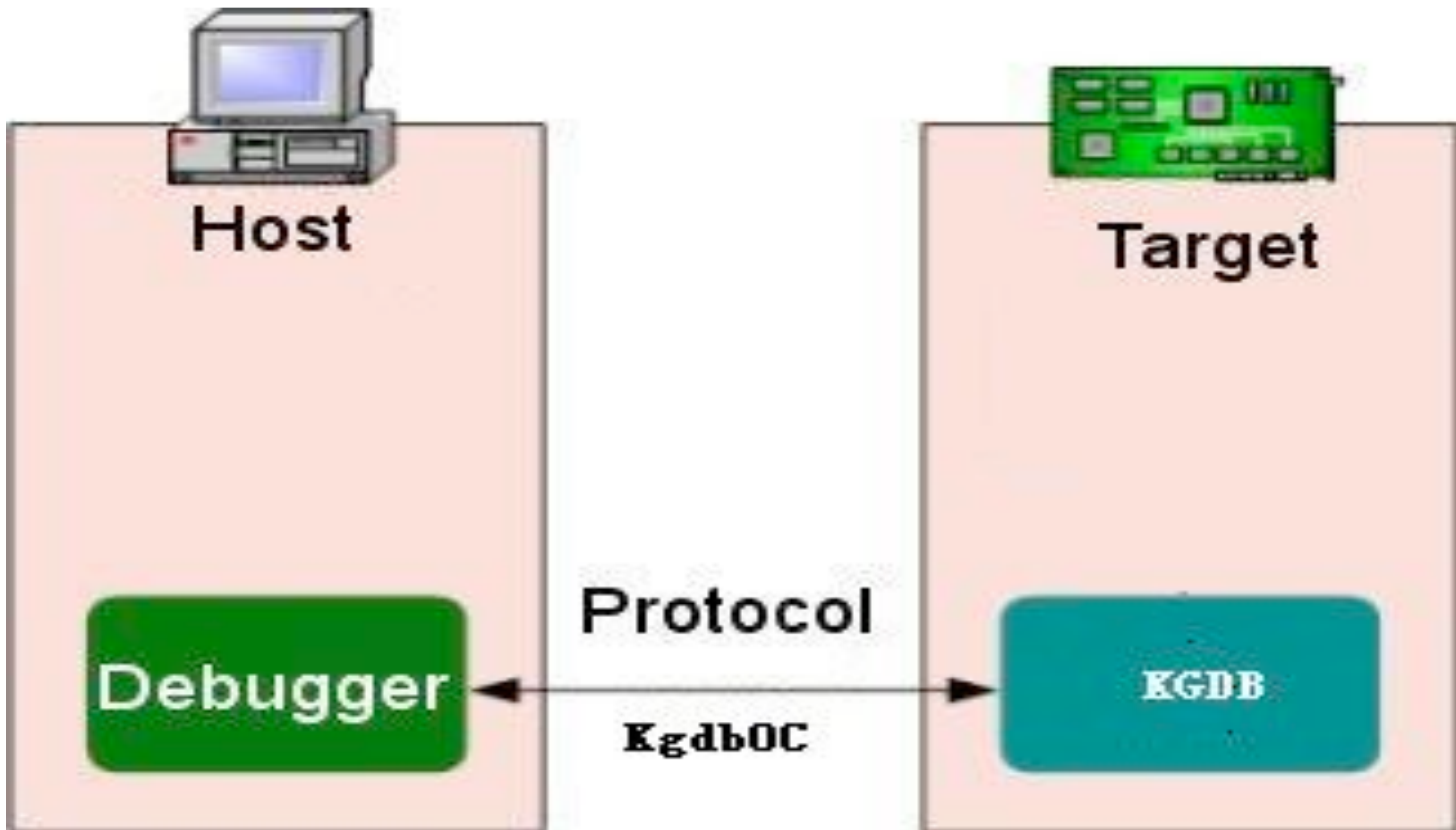
Debugger

- How debugger works

- Interrupt
  - hardware interrupt
  - exception ---->debug exception
  - software interrupt

- Key components of debugger
  - take over the debug exception
  - pick and poke system info (registers, memory)
  - communicable ----> could receive and deliver data with others

KGDB



KGDB.info

KGDB using

- KGDB was merged to kernel since 2.6.28
- KGDB Config make menuconfig
  - **CONFIG_KGDB**
  - **CONFIG_KGDB_SERIAL_CONSOLE**
  - **CONFIG_DEBUG_INFO**
  - **CONFIG_FRAME_POINTER**
  - **CONFIG_MAGIC_SYSRQ**
  - **CONFIG_DEBUG_RODATA = n**

KGDB using

- Kgdboc
  - **build in kernel**

  echo "ttyS0,115200" >/sys/module/kgdboc/parameters/kgdboc
  - **module**

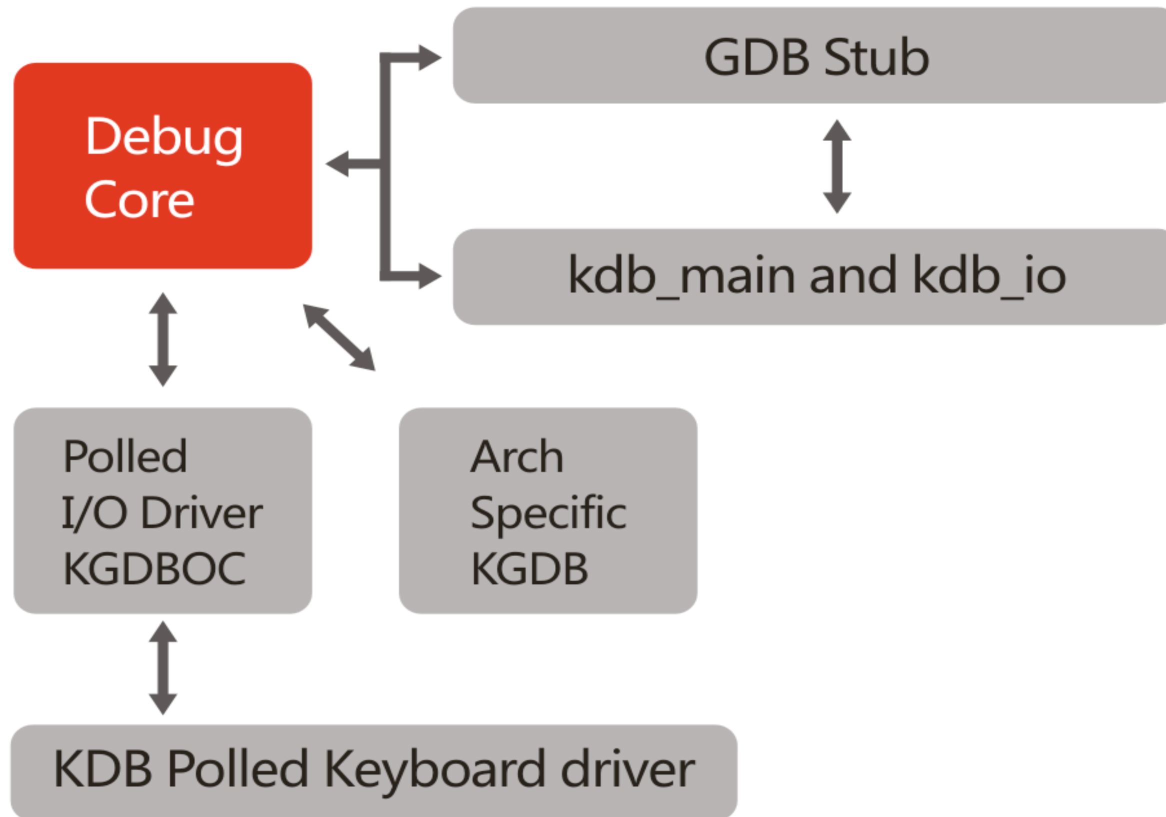
  Insmod kgdboc.ko kgdboc="ttyS0,115200"
- Gdb
  - **gdb /usr/src/work/vmlinux**
  - (gdb) **set remotebaud 115200**
  - (gdb) **target remote /dev/ttyS0**

  Remote debugging using /dev/ttyS0
  kgdb_breakpoint () at kernel/debug/debug_core.c:983
  983 wmb(); /* Sync point after breakpoint */
  (gdb)
- Trap to kgdb by magic key ----> **echo "g" >/proc/sysrq-trigger**

KGDB using

- Gdb
  - (gdb) **b address/functions**
  - (gdb) **s / si / n / c**
  - (gdb) **bt**
  - (gdb) **info register/break/threads**
  - (gdb) **watch/rwatch (currently only x86 support)**
  - (gdb) **m addr**
  - (gdb) **set val=abc**
  - (gdb) **l* function+0x16**

KGDB arch



KGDB.info

Unoptimized debugging

- un-optimize single file

  CFLAGS_filename.o += -O0

- un-optimize entire directory of files

  EXTRA_CFLAGS += -O0

- un-optimize kernel module
  - make -C build linux.modules COPTIMIZE=-O0 M=path_to_source

- DO NOT UN_OPTIMEZ the whole kernel
  - some codes were hacked as compiler specific.

Got a timing problem? Use variables

- Use a conditional variable to control a printk()
  - If (dbg_con) { printk("state info ..."); }
- Use a conditional to execute a variable++
  - If (dbg_con) { var++; }
- Use a conditional to execute a function
  - If (dbg_con) { xxx_function(); }
- Debugger set conditional counter
  - (gdb) set dbg_con=1

Questions of Debugger

- How kernel debugger works on multi-cpus (SMP)
  - Before enter debugger core route ---
    hold on the others slave cpu through IPI

  - Before quit debugger route ---
    release the slave cpus
    (tips: run flag → atomic variable, spinlock, row_spinlock)

- How kernel debugger works on multi-processes
  - Have problems?
    single step on specified process,
    schedule

- Other debugger questions?

Crash Analyse

- Where are the crash coming
  - BUG
  - Oops
  - Panic

- Other info
  - Linux/Documentation/oops-tracing.txt

Crash Analyse

```
BUG: unable to handle kernel NULL pointer dereference at   (null)
    IP: [<c01683c7>] proc_dowatchdog+0x7/0xd0
    *pde = 00000000
    Oops: 0002 [#2] PREEMPT
    Modules linked in:
    Pid: 1126, comm: sh Tainted: G      D     3.0.0-rc2-dirty #4 Bochs Bochs
    EIP: 0060:[<c01683c7>] EFLAGS: 00000286 CPU: 0 → Register Info
    EIP is at proc_dowatchdog+0x7/0xd0
    EAX: c069fcc4 EBX: 00000001 ECX: b7838000 EDX: 00000001
    ESI: c069fcc4 EDI: 00000004 EBP: b7838000 ESP: d7623f30
     DS: 007b ES: 007b FS: 0000 GS: 0033 SS: 0068
    Process sh (pid: 1126, ti=d7622000 task=d749f4a0 task.ti=d7622000)
    Stack:
     d749f4a0 c069f6c0 c069f6c0 c069fcc4 c0202c77 d7623f50 d7623f9c 00000000
     00000004 d7623f9c 00000004 b7838000 c0202cb0 c0202cc8 d7623f9c 00000001
     d7428e00 c01b4d50 d7623f9c 00000002 00000001 d7428e00 fffffff7 081d1300
    Call Trace:
     [<c0202c77>] ? proc_sys_call_handler+0x77/0xb0
     [<c0202cb0>] ? proc_sys_call_handler+0xb0/0xb0
     [<c0202cc8>] ? proc_sys_write+0x18/0x20
     [<c01b4d50>] ? vfs_write+0xa0/0x140
     [<c01b4ec1>] ? sys_write+0x41/0x80
     [<c0539d10>] ? sysenter_do_call+0x12/0x26
    Code: 75 0f c7 03 01 00 00 00 e8 57 69 fd ff 85 c0 74 db a1 48 c0 69
    c0 c7 00 00 00 00 00 31 c0 83 c4 04 5b c3 90 56 53 89 d3 83 ec 08 <c7>
    05 00 00 00 00 05 00 00 00 8b 54 24 18 89 54 24 04 8b 54 24
    EIP: [<c01683c7>] proc_dowatchdog+0x7/0xd0 SS:ESP 0068:d7623f30
    CR2: 0000000000000000
    ---[ end trace 8b37721a29dead5b ]--
```

KGDB.info

Crash Analyse

```
(gdb) l* proc_dowatchdog+0x7
0xc01683c7 is in proc_dowatchdog (kernel/watchdog.c:522).
517                     void __user *buffer, size_t *lenp, loff_t *ppos)
518     {
519             int ret;
520
521             int* xx = NULL;
522             *xx = 5;
523
524             ret = proc_dointvec_minmax(table, write, buffer, lenp, ppos);
525             if (ret || !write)
526                     goto out;
(gdb)
```

Debugging process

- Reproduce problem
  - find/read all related documents of problem
  - version back up/go forward
  - reduce dependence

- Analyse problem
  - do more experiments, no guess !!!

- Fix problem
  - got real root cause?
  - patch --- simple, clear
  - enjoy and play kernel!

Debugging tricks

Kernel Hacking config

- Get debugging information in case of kernel bugs
  - CONFIG_FRAME_POINTER
- Lockup (soft/hard) detector
  - CONFIG_LOCKUP_DETECTOR
- SpinLock detector
  - CONFIG_DEBUG_SPINLOCK
- RCU cpu stall detector
  - CONFIG_RCU_CPU_STALL_DETECTOR
- softlockup / time interrupt
  - check hang system
  - soft watchdog
  - softlockup.c : softlockup_tick()
- NMI
  - check hang system
  - hardware watchdog: nmi_watchdog=1

Print Functions

- Some useful Print function for development
  - BUG_ON()
  - WARN_ON
  - show_backtrace()
  - panic()
  - die()
  - show_registers()
  - print_symbol(pointer)
  - get function caller:

  return_address() → gcc __builtin_return_address(0)

Thanks

Feedback to: libfetion@gmail.com

Or visiting: http://www.kgdb.info