

如何利用硬件watchpoint定位踩内存问题

2019-07-21 coderhuo 操作系统 arm RTOS watchpoint Hardware Breakpoint 踩内存

本文介绍如何使用ARM平台的硬件watchpoint定位踩内存问题，特别是如何在运行过程中自动对特定内存区域添加watchpoint。

在踩内存问题中，最困难的就是找出元凶。常见的作法如下：

- 1. 通过gdb打内存断点（添加watchpoint），看看谁非法访问了该内存区域。本方法的局限性在于：有些系统不支持gdb，或者被踩内存地址不固定，或者问题出现在启动阶段，来不及设置断点。
- 2. 通过MMU（Linux下可以使用mprotect）对特定内存区域进行保护。本方法的局限性在于：MMU保护的最小单位是一个内存页（一般为4KB），有可能受害内存区域较小，无法用MMU进行保护。
- 3. dump事发现场周边的内存，通过关键字识别谁对这块内存进行了非法写入。比如受害内存区域中有0xAABB字样，而只有某个模块会产生0xAABB的数据，基于此就可以锁定凶手。但是并非每个模块的数据都是有特征的，大部分情况下无法通过该方法找到凶手。

这时可以尝试芯片自带的**硬件watchpoint功能**，ARM平台和x86/64一般均支持。其实gdb的watchpoint大多数情况下就是基于硬件中断实现的（<https://sourceware.org/gdb/wiki/Internals%20Watchpoints>）。

下面基于ARM Cortex-A9介绍如何使用该功能，本文大量引用《ARM® Cortex®-A9 Technical Reference Manual I》、《ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition》和hw_breakpoint.c中的内容。

一、首先确认是否支持硬件watchpoint

这个必须查对应芯片的技术手册。从《ARM® Cortex®-A9 Technical Reference Manual》10.3.2节（Breakpoints and watchpoints）可以看到，本处理器支持6个breakpoints、4个watchpoints。

二、打开监控模式

本节的原理可以参考《ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition》C11.11.20 DBGDSCR, Debug Status and Control Register 这一节。

本节所展示代码均摘自或基于hw_breakpoint.c中相关代码改写。

要想使用硬件watchpoint，必须先打开监控模式。下面的代码可以判断当前是否已经打开监控模式：

```
/*
 * In order to access the breakpoint/watchpoint control registers,
 * we must be running in debug monitor mode.
 */
static int monitor_mode_enabled(void)
{
    u32 dscr;
    ARM_DBG_READ(c0, c1, 0, dscr);
    return !(dscr & ARM_DSCR_MDBGEN);
}
```

其中寄存器读写操作的宏定义如下：

```
#define ARM_DSCR_HDBGEN        (1 << 14) /* DSCR halting bits. */
#define ARM_DSCR_MDBGEN        (1 << 15) /* DSCR monitor bits. */

/* Accessor macros for the debug registers. */
#define ARM_DBG_READ(N, M, OP2, VAL) do {\
    asm volatile("mrc p14, 0, %0, " #N ", " #M ", " #OP2 : "=r" (VAL));\
} while (0)
```

Content

- 一、首先确认是否支持硬件watchpoint
- 二、打开监控模式
- 三、设置watchpoint
- 四、参考资料
- Similar Posts
- Comments

```
#define ARM_DBG_WRITE(N, M, OP2, VAL) do {\
    asm volatile("mcr p14, 0, %0, " #N ", " #M ", " #OP2 : : "r" (VAL));\
} while (0)
```

下面的代码可以打开监控模式：

```
static int enable_monitor_mode(void)
{
    u32 dscr;
    ARM_DBG_READ(c0, c1, 0, dscr);

    /* If monitor mode is already enabled, just return. */
    if (dscr & ARM_DSCR_MDBGEN)
        goto out;

    /* Write to the corresponding DSCR. */
    /* 根据不同的平台, 调用不同的代码打开监控模式 */
    switch (get_debug_arch()) {
    case ARM_DEBUG_ARCH_V6:
    case ARM_DEBUG_ARCH_V6_1:
        ARM_DBG_WRITE(c0, c1, 0, (dscr | ARM_DSCR_MDBGEN));
        break;
    case ARM_DEBUG_ARCH_V7_ECP14:
    case ARM_DEBUG_ARCH_V7_1:
    case ARM_DEBUG_ARCH_V8:
        ARM_DBG_WRITE(c0, c2, 2, (dscr | ARM_DSCR_MDBGEN));
        isb();
        break;
    default:
        return -ENODEV;
    }

    /* Check that the write made it through. */
    /* 检查是否已经打开监控模式 */
    ARM_DBG_READ(c0, c1, 0, dscr);
    if (!(dscr & ARM_DSCR_MDBGEN)) {
        pr_warn_once("Failed to enable monitor mode on CPU %d.\n",
                     smp_processor_id());
        return -EPERM;
    }

out:
    return 0;
}
```

三、设置watchpoint

本节的原理可以参考《ARM® Cortex®-A9 Technical Reference Manual》10.5.3 (Watchpoint Value Registers)和10.5.4(Watchpoint Control Registers) 这两节。

本节所展示代码均摘自或基于hw_breakpoint.c中相关代码改写。

硬件watchpoint功能，是由Watchpoint Value Register (WVR) 和Watchpoint Control Register(WCR)两个寄存器配对实现的，前者设置被监控地址（虚拟内存地址，而不是物理内存地址，这省去了转换环节，极大的方便了调试），后者进行控制。

下面的代码可以用来设置Watchpoint，它的作用是：如果有人在用户态往addr开始的前两个字节写入内容，就会产生异常。

```
/*
 * i: watchpoint寄存器序号, 对于ARM-A9, 可用寄存器为c0~c3, i 取值范围为0~3
 * addr: 必须是虚拟内存地址, 且必须是字对齐的(32位系统为4字节对齐)
 * we must be running in debug monitor mode.
```

```

*/

int arm_install_hw_watchpoint(int i, u32 addr)
{
    u32 ctrl = 0x117;
    u32 check_value;

    /* Setup the address register. */
    write_wb_reg(ARM_BASE_WVR + i, addr);
    check_value = read_wb_reg(ARM_BASE_WVR + i);
    if (check_value != addr)
    {
        pr_warn("fail to set WVR[%d] addr:0x%x check_value:0x%x\n", i, addr, check_value);
        return -1;
    }

    /* Setup the control register. */
    write_wb_reg(ARM_BASE_WCR + i, ctrl);
    check_value = read_wb_reg(ARM_BASE_WCR + i);
    if (check_value != ctrl)
    {
        pr_warn("fail to set WCR[%d] ctrl:0x%x check_value:0x%x\n", i, ctrl, check_value);
        return -1;
    }

    return 0;
}

```

其中ctrl = 0x117 (二进制0011 10 10 1)的解释如下, 主要参考《ARM® Cortex®-A9 Technical Reference Manual》10.5.4(Watchpoint Control Registers) 这一节：

1. 最低位的1表示开启Watchpoint功能
2. 再向上的10代表仅监控用户模式下对该内存区域的操作
3. 再向上的10代表仅监控往该内存区域的写入操作
4. Watchpoint监控的虚拟内存地址为字对齐的（32位系统为4字节对齐），每个Watchpoint最大监控长度为4字节（32位系统）。但是如果仅仅想监控1个字节或者2个字节呢？最前面的四个比特0011就是用来做这个事情的。上面的0011代表仅监控addr开始的2个字节。如果只想监控最后一个字节，前4比特可以写为0001。

上面代码中引用函数/宏定义如下：

```

#define isb() __asm__ __volatile__ (" : : : \"memory\")

#define READ_WB_REG_CASE(OP2, M, VAL) \
    case ((OP2 << 4) + M): \
        ARM_DBG_READ(c0, c ## M, OP2, VAL); \
        break

#define WRITE_WB_REG_CASE(OP2, M, VAL) \
    case ((OP2 << 4) + M): \
        ARM_DBG_WRITE(c0, c ## M, OP2, VAL); \
        break

#define GEN_READ_WB_REG_CASES(OP2, VAL) \
    READ_WB_REG_CASE(OP2, 0, VAL); \
    READ_WB_REG_CASE(OP2, 1, VAL); \
    READ_WB_REG_CASE(OP2, 2, VAL); \
    READ_WB_REG_CASE(OP2, 3, VAL); \
    READ_WB_REG_CASE(OP2, 4, VAL); \
    READ_WB_REG_CASE(OP2, 5, VAL); \
    READ_WB_REG_CASE(OP2, 6, VAL); \
    READ_WB_REG_CASE(OP2, 7, VAL); \
    READ_WB_REG_CASE(OP2, 8, VAL); \
    READ_WB_REG_CASE(OP2, 9, VAL); \

```

```

    READ_WB_REG_CASE(OP2, 10, VAL);      \
    READ_WB_REG_CASE(OP2, 11, VAL);      \
    READ_WB_REG_CASE(OP2, 12, VAL);      \
    READ_WB_REG_CASE(OP2, 13, VAL);      \
    READ_WB_REG_CASE(OP2, 14, VAL);      \
    READ_WB_REG_CASE(OP2, 15, VAL)

#define GEN_WRITE_WB_REG_CASES(OP2, VAL) \
    WRITE_WB_REG_CASE(OP2, 0, VAL);      \
    WRITE_WB_REG_CASE(OP2, 1, VAL);      \
    WRITE_WB_REG_CASE(OP2, 2, VAL);      \
    WRITE_WB_REG_CASE(OP2, 3, VAL);      \
    WRITE_WB_REG_CASE(OP2, 4, VAL);      \
    WRITE_WB_REG_CASE(OP2, 5, VAL);      \
    WRITE_WB_REG_CASE(OP2, 6, VAL);      \
    WRITE_WB_REG_CASE(OP2, 7, VAL);      \
    WRITE_WB_REG_CASE(OP2, 8, VAL);      \
    WRITE_WB_REG_CASE(OP2, 9, VAL);      \
    WRITE_WB_REG_CASE(OP2, 10, VAL);     \
    WRITE_WB_REG_CASE(OP2, 11, VAL);     \
    WRITE_WB_REG_CASE(OP2, 12, VAL);     \
    WRITE_WB_REG_CASE(OP2, 13, VAL);     \
    WRITE_WB_REG_CASE(OP2, 14, VAL);     \
    WRITE_WB_REG_CASE(OP2, 15, VAL)

static u32 read_wb_reg(int n)
{
    u32 val = 0;

    switch (n) {
        GEN_READ_WB_REG_CASES(ARM_OP2_BVR, val);
        GEN_READ_WB_REG_CASES(ARM_OP2_BCR, val);
        GEN_READ_WB_REG_CASES(ARM_OP2_WVR, val);
        GEN_READ_WB_REG_CASES(ARM_OP2_WCR, val);
        default:
            pr_warn("attempt to read from unknown breakpoint register %d\n", n);
    }

    return val;
}

static void write_wb_reg(int n, u32 val)
{
    switch (n) {
        GEN_WRITE_WB_REG_CASES(ARM_OP2_BVR, val);
        GEN_WRITE_WB_REG_CASES(ARM_OP2_BCR, val);
        GEN_WRITE_WB_REG_CASES(ARM_OP2_WVR, val);
        GEN_WRITE_WB_REG_CASES(ARM_OP2_WCR, val);
        default:
            pr_warn("attempt to write to unknown breakpoint register %d\n", n);
    }

    isb();
}

```

注意：cache操作不会产生watchpoint事件(Cache maintenance operations do not generate watchpoint events)

有了上面的`arm_install_hw_watchpoint`函数，我们就可以根据需要在程序运行过程中动态的对特定地址添加监控，看看谁踩了内存。

四、参考资料

1. 《ARM® Cortex®-A9 Technical Reference Manual》
2. 《ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition》
3. <https://sourceware.org/gdb/wiki/Internals%20Watchpoints>
4. hw_breakpoint.c

Similar Posts

- Cache一致性导致的踩内存问题
- 如何在实时操作系统(RTOS)中使用GCC的栈溢出保护(SSP)功能
- Ubuntu上搭建arm虚拟运行环境

上一篇 [如何在实时操作系统\(RTOS\)中使用GCC的栈溢出保护\(SSP\)功能](#)

下一篇 [Cache一致性导致的踩内存问题](#)

Comments

0 comments

Anonymous ▾




Leave a comment

① Markdown is supported

Login with GitHub

Preview

Be the first guy leaving a comment!

Contact me at:  

本站总访问量28852次，本文总阅读量288次

Site powered by Jekyll & Github Pages. Theme designed by HyG.
