64 位多核 MIPS 异常和中断内核代码分析 (1)

1. 高优先级例外入口初始化

本文不涉及 Bootloader，因此 Reset 例外入口处的 Bootloader 我们就不讨论了。

多核环境下，所有核默认指向的例外入口基地址都是一样的，尽管可以修改各自核内部的 EBase 寄存器来改变例外入口基地址，但我们此处只讨论所有核运行在对称多处理模式 (SMP) 下的情形，即所有核都使用同样的例外入口

1.1 Cache Error

先看看 Cache Error 时，Linux Kernel 都做了什么

1.1.1 Loongson 2E Cache Error

Cache Error 例外入口初始化，位于：

```
[arch/mips/mm/c-r4k.c]
void __init r4k_cache_init()
{
.....
set_uncached_handler(0x100, &except_vec2_generic, 0x80);
.....
}
```

因为 cache 错误时可以 cache 的 KSEG0 段不能用了，则 cache 错误例外处理函数位于 KSEG1 之0xFFFF FFFF A000 0000 + 0x100 处，长度最大为128 Bytes（32 条指令），处理函数为 except_vec2_generic，定义于：

```
[arch/mips/mm/cex-gen.S]
/*
 * Game over. Go to the button. Press gently. Swear where allowed by
 * legislation.
 */
 LEAF(except_vec2_generic)
 .set noreorder
 .set noat
 .set mips0
 /*
 * This is a very bad place to be. Our cache error
 * detection has triggered. If we have write-back data
 * in the cache, we may not be able to recover. As a
 * first-order desperate measure, turn off KSEG0 cacheing.
 */
 mfc0 k0,CP0_CONFIG
 li k1,~CONF_CM_CMASK
 and k0,k0,k1
 ori k0,k0,CONF_CM_UNCACHED # 改变 KSEG0 为 Uncached
 mtc0 k0,CP0_CONFIG
 /* Give it a few cycles to sink in... */
```

```
  nop
  nop
  nop

  j cache_parity_error
  nop
  END(except_vec2_generic)
```

发生 Cache Error 后，Cache 已不再可信，因此原 KSEG0 不能再使用 Cache，首先改变其为 Uncached，然后进入真正的处理函数 cache_parity_error：

```
[arch/mips/kernel/traps.c]
asmlinkage void cache_parity_error(void)
{
  const int field = 2 * sizeof(unsigned long);
  unsigned int reg_val;

  /* For the moment, report the problem and hang. */
  printk("Cache error exception:\n");
  printk("cp0_errorepc == %0*lx\n", field, read_c0_errorepc());
  reg_val = read_c0_cacheerr();
  printk("c0_cacheerr == %08x\n", reg_val);

  printk("Decoded c0_cacheerr: %s cache fault in %s reference.\n",
  reg_val & (1<<30) ? "secondary" : "primary",
  reg_val & (1<<31) ? "data" : "insn");
  printk("Error bits: %s%s%s%s%s%s%s\n",
  reg_val & (1<<29) ? "ED " : "",
  reg_val & (1<<28) ? "ET " : "",
  reg_val & (1<<26) ? "EE " : "",
  reg_val & (1<<25) ? "EB " : "",
  reg_val & (1<<24) ? "EI " : "",
  reg_val & (1<<23) ? "E1 " : "",
  reg_val & (1<<22) ? "E0 " : "");
  printk("IDX: 0x%08x\n", reg_val & ((1<<22)-1));

#if defined(CONFIG_CPU_MIPS32) || defined(CONFIG_CPU_MIPS64)
  if (reg_val & (1<<22))
  printk("DErrAddr0: 0x%0*lx\n", field, read_c0_derraddr0());

  if (reg_val & (1<<23))
  printk("DErrAddr1: 0x%0*lx\n", field, read_c0_derraddr1());
#endif
  panic("Can't handle the cache error!");
}
```

事实上，当出现 Cache Error 时，系统已经没法修复，只能打出一些可供判断的信息后 panic :(
另外 set_uncached_handler 定义于：

[arch/mips/kernel/traps.c]

```
/*
 * Install uncached CPU exception handler.
 * This is suitable only for the cache error exception which is the only
 * exception handler that is being run uncached.
 */
void __cpuinit set_uncached_handler(unsigned long offset, void *addr,
 unsigned long size)
{
 unsigned long uncached_ebase = CKSEG1ADDR(ebase);

 if (!addr)
 panic(panic_null_cerr);

 memcpy((void *)(uncached_ebase + offset), addr, size);
}
```

其中 CKSEG1ADDR 的宏用于将虚址 ebase 转化为对应的 uncached 段的虚址 0xFFFF FFFF A000 0000。可以看到这个函数负责把例外处理函数 except_vec2_generic 复制到 Cache Error 例外的入口 0xFFFF FFFF A000 0100 处，即物理地址 0x100 处

1.1.2 Cavium Octeon Cache Error

Cavium Octeon 的 Cache 错误处理和 2E 大同小异，只是处理过程多了些鲁棒性
其 Cache Error 例外入口初始化，位于：

[arch/mips/mm/c-octeon.c]

```
void __cpuinit octeon_cache_init(void)
{
 extern unsigned long ebase;
 extern char except_vec2_octeon;

 memcpy((void *)(ebase + 0x100), &except_vec2_octeon, 0x80);
 octeon_flush_cache_sigtramp(ebase + 0x100);
.....
}
```

将例外处理函数 except_vec2_octeon 复制到 ebase + 0x100 处，在 Cavium 上 ebase 为：

[arch/mips/cavium-octeon/setup.c]

```
uint32_t ebase = read_c0_ebase() & 0x3ffff000
```

实际就是 EBase 寄存器中的值，默认情形下，ebase = 0x8000 0000，ebase + 0x100 就是

0x8000 0180，即物理地址 0x100 处。因为使用 KSEG0 的虚拟地址访问，数据会被缓存，因此为了及时生效到内存，还要 flush 一下 cache。
另外在 64 位的处理器上使用 32 位的地址访问，处理器会自动将其扩展为 64 位，规则就是 32位高位符号扩展。0x8000 0180 最高位为 1，则会被扩展为 0xFFFF FFFF 8000 0180；0x0000 0180 则会被扩展为 0x0000 0000 0000 0180
其中，Octeon 的 Cache Error 例外处理函数 except_vec2_octeon 定义于：
[arch/mips/mm/cex-oct.S]

```
/*
 * Handle cache error. Indicate to the second level handler whether
 * the exception is recoverable.
 */
 LEAF(except_vec2_octeon)

 /* due to an errata we need to read the COP0 CacheErr (Dcache)
 * before any cache/DRAM access */

 rdhwr k0, $0 /* get core_id */
 PTR_LA k1, cache_err_dcache
 sll k0, k0, 3
 PTR_ADDU k1, k0, k1 /* k1 = &cache_err_dcache[core_id] */

 dmfc0 k0, CP0_CACHEERR, 1
 sd k0, (k1)
 dmtc0 $0, CP0_CACHEERR, 1 # 把 DCache_Error 寄存器的值存入 cache_err_dcache[core_id] 数组

 /* check whether this is a nested exception */
 mfc0 k1, CP0_STATUS
 andi k1, k1, ST0_EXL
 beqz k1, 1f
 nop
 j cache_parity_error_octeon_non_recoverable
 nop


/* exception is recoverable */
1: j handle_cache_err
 nop

 END(except_vec2_octeon)
```

回顾一下，例外发生时 CPU 会将 STATUS[EXL] 置为 1，但是在 Cache Error 例外出现时，EXL 不会被置为 1，CPU 而是将 STATUS[ERL] 置为 1，这个意味着：
I. CPU 自动进入内核模式（忽略 STATUS[KSU] 位），禁止中断（忽略 STATUS[IE] 位）
II. eret 指令使用 CP0_ErrorEPC 代替 CP0_EPC 作为例外返回地址
III. Kuseg 和 xkuseg 被改为 unmapped, uncached 的区域，以便在 cache 不能用时，内存

空间能正常访问
貌似可恢复 Cache Error 的处理过程如下：
访问地址 VA ---> 第一次发生 Cache Error 例外 ---> 进入 except_vec2_octeon，STATUS[ERL] = 1 ---> CP0_STATUS & ST0_EXL 为 0 ---> 进入

handle_cache_err---> SAVE_ALL 保存上下文后，KMODE 清除 ERL 位 ---> 进入貌似 Cache Error 可恢复的处理函数 cache_parity_error_octeon_recoverable ---> 打出 CP0_ICache_Error/CP0_DCache_Error 和 CP0_ErrorEPC ---> 执行 eret 指令，例外返回到 ErrorEPC 指向的地址，即刚刚访问出错的地址 VA 再试一下
错误不可恢复的情形：
嵌套异常，即在已有的异常处理过程中，EXL 还没有被清除，却发生了 Cache Error，此时进入 except_vec2_octeon，STATUS[ERL] = 1 --->CP0_STATUS & ST0_EXL 为 1 ---> 最终进入 cache_parity_error_octeon_non_recoverable，打印 CP0_ICache_Error/CP0_DCache_Error 和 CP0_ErrorEPC 后，直接 panic

```
 /* We need to jump to handle_cache_err so that the previous handler
 * can fit within 0x80 bytes. We also move from 0xFFFFFFFFAXXXXXXX
 * space (uncached) to the 0xFFFFFFFF8XXXXXXX space (cached). */
 LEAF(handle_cache_err)
 .set push
 .set noreorder
 .set noat

 SAVE_ALL
 KMODE # clear EXL, ERL, set kernel mode bit
 jal cache_parity_error_octeon_recoverable
 nop
 j ret_from_exception
 nop

 .set pop
 END(handle_cache_err)
[arch/mips/mm/c-octeon.c]
asmlinkage void cache_parity_error_octeon_recoverable(void)
{
 cache_parity_error_octeon(0);
}

asmlinkage void cache_parity_error_octeon_non_recoverable(void)
{
 cache_parity_error_octeon(1);
}
static void cache_parity_error_octeon(int non_recoverable)
{
```

```
  unsigned long coreid = cvmx_get_core_num();
  uint64_t icache_err = read_octeon_c0_icacheerr();

  pr_err("Cache error exception:\n");
  pr_err("cp0_errorepc == %lx\n", read_c0_errorepc());
  if (icache_err & 1) {
  pr_err("CacheErr (Icache) == %llx\n",
  (unsigned long long)icache_err);
  write_octeon_c0_icacheerr(0);
  }
  if (cache_err_dcache[coreid] & 1) {
  pr_err("CacheErr (Dcache) == %llx\n",
  (unsigned long long)cache_err_dcache[coreid]);
  cache_err_dcache[coreid] = 0;
  }

  if (non_recoverable)
  panic("Can't handle cache error: nested exception");
}
```