支持该feature的code在

arch/arm/nwfpe/fpmodule.c

arch/arm/kernel/traps.c

arch/arm/mm/fault.c

arch/arm/kernel/traps.c

```
#ifdef CONFIG_DEBUG_USER
unsigned int user_debug;

static int __init user_debug_setup(char *str)
{
    get_option(&str, &user_debug);
    return 1;
}
__setup("user_debug=", user_debug_setup);
#endif
```

kernel boot parameter "user_debug=xx" 记录在user_debug variable中。

```
asmlinkage void __exception do_undefinstr(struct pt_regs *regs)
{
    ......

die_sig:
#ifdef CONFIG_DEBUG_USER
    if (user_debug & UDBG_UNDEFINED) {
        printk(KERN_INFO "%s (%d): undefined instruction: pc=%p\n",
            current->comm, task_pid_nr(current), pc);
        __show_regs(regs);
        dump_instr(KERN_INFO, regs);
    }
#endif

    info.si_signo = SIGILL;
    info.si_errno = 0;
    info.si_code  = ILL_ILLOPC;
    info.si_addr  = pc;

    arm_notify_die("Oops - undefined instruction", regs, &info, 0, 6);
}
```

在执行到非法指令时，dump出详细信息

```
static int bad_syscall(int n, struct pt_regs *regs)
{
    ......
#ifdef CONFIG_DEBUG_USER
    if (user_debug & UDBG_SYSCALL) {
        printk(KERN_ERR "[%d] %s: obsolete system call %08x.\n",
            task_pid_nr(current), current->comm, n);
        dump_instr(KERN_ERR, regs);
    }
#endif

    info.si_signo = SIGILL;
    info.si_errno = 0;
    info.si_code  = ILL_ILLTRP;
    info.si_addr  = (void __user *)instruction_pointer(regs) -
             (thumb_mode(regs) ? 2 : 4);

    arm_notify_die("Oops - bad syscall", regs, &info, n, 0);

    return regs->ARM_r0;
}
```

在trigger bad syscall后同样输出详细信息。

```
/*
 * Handle all unrecognised system calls.
 *  0x9f0000 - 0x9fffff are some more esoteric system calls
 */
#define NR(x) ((__ARM_NR_##x) - __ARM_NR_BASE)
asmlinkage int arm_syscall(int no, struct pt_regs *regs)
{
    ......

#ifdef CONFIG_DEBUG_USER
    /*
     * experience shows that these seem to indicate that
     * something catastrophic has happened
     */
    if (user_debug & UDBG_SYSCALL) {
        printk("[%d] %s: arm syscall %d\n",
               task_pid_nr(current), current->comm, no);
        dump_instr("", regs);
        if (user_mode(regs)) {
            __show_regs(regs);
            c_backtrace(frame_pointer(regs), processor_mode(regs));
        }
    }
#endif
    info.si_signo = SIGILL;
    info.si_errno = 0;
    info.si_code  = ILL_ILLTRP;
    info.si_addr  = (void __user *)instruction_pointer(regs) -
             (thumb_mode(regs) ? 2 : 4);

    arm_notify_die("Oops - bad syscall(2)", regs, &info, no, 0);
    return 0;
}
```

```
/*
 * A data abort trap was taken, but we did not handle the instruction.
 * Try to abort the user program, or panic if it was the kernel.
 */
asmlinkage void
baddataabort(int code, unsigned long instr, struct pt_regs *regs)
{
    unsigned long addr = instruction_pointer(regs);
    siginfo_t info;

#ifdef CONFIG_DEBUG_USER
    if (user_debug & UDBG_BADABORT) {
        printk(KERN_ERR "[%d] %s: bad data abort: code %d instr 0x%08lx\n",
            task_pid_nr(current), current->comm, code, instr);
        dump_instr(KERN_ERR, regs);
        show_pte(current->mm, addr);
    }
#endif

    info.si_signo = SIGILL;
    info.si_errno = 0;
    info.si_code  = ILL_ILLOPC;
    info.si_addr  = (void __user *)addr;

    arm_notify_die("unknown data abort code", regs, &info, instr, 0);
}
```

都是在出错时,kernel输出application的信息。

其中最关键的是dump_instr()

```
static void dump_instr(const char *lvl, struct pt_regs *regs)
{
    unsigned long addr = instruction_pointer(regs);     ①
    const int thumb = thumb_mode(regs);
    const int width = thumb ? 4 : 8;
    mm_segment_t fs;
    char str[sizeof("00000000 ") * 5 + 2 + 1], *p = str;
    int i;

    /*
     * We need to switch to kernel mode so that we can use __get_user
     * to safely read from kernel space.  Note that we now dump the
     * code first, just in case the backtrace kills us.
     */
    fs = get_fs();
    set_fs(KERNEL_DS);

    for (i = -4; i < 1 + !!thumb; i++) {            ②
        unsigned int val, bad;

        if (thumb)
            bad = __get_user(val, &((u16 *)addr)[i]);
        else
            bad = __get_user(val, &((u32 *)addr)[i]);       ③

        if (!bad)                                           ④
            p += sprintf(p, i == 0 ? "(%0*x) " : "%0*x ",   ⑤
                    width, val);
        else {                                              ⑥
            p += sprintf(p, "bad PC value");
            break;
        }
    }
    printk("%sCode: %s\n", lvl, str);

    set_fs(fs);
}
```

struct pt_regs *regs是application出错时的register dump。

①

```
#define instruction_pointer(regs)   (regs)->ARM_pc
```

取得引起该trap的application正执行的指令地址

②
输出出错指令前 4 条指令

③
从application的address space读取指令，返回 0 表示能读取application的指令，否则application的address

space已经被破坏

④⑤
输出指令

⑥
application的地址空间已经被破坏