在menuconfig中对CONFIG_DEBUG_USER有如下描述

CONFIG_DEBUG_USER:

When a user program crashes due to an exception, the kernel can print a brief message explaining what the problem was. This is sometimes helpful for debugging but serves no purpose on a production system. Most people should say N here.

In addition, you need to pass user_debug=N on the kernel command line to enable this feature. N consists of the sum of:

1 - undefined instruction events

2 - system calls

4 - invalid data aborts

8 - SIGSEGV faults

16 - SIGBUS faults

Symbol: DEBUG_USER [=n]

Type : boolean

Prompt: Verbose user fault messages

Defined at arch/arm/Kconfig.debug:49

Location:

-> Kernel hacking

在CONFIG_DEBUG_USER=y 的情况下对用户态application crash后调试有利！

How to enable "user_debug" feature ?

1. CONFIG_DEBUG_USER=y

2. add user_debug=X on kernel parameter

in arch/arm/include/asm/system_misc.h

```
#define UDBG_UNDEFINED  (1 << 0)
#define UDBG_SYSCALL    (1 << 1)
#define UDBG_BADABORT   (1 << 2)
#define UDBG_SEGV       (1 << 3)
#define UDBG_BUS        (1 << 4)

extern unsigned int user_debug;
```

即user_debug可以控制打开哪些"user_debug" feature

user_debug=0x1f

打开所有feature

user_debug=1

只打开非法指令

---

kernel中对user_debug的支持

in 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
```

1. 对非法指令的支持

```
asmlinkage void __exception do_undefinstr(struct pt_regs *regs)
{
        unsigned int instr;
        siginfo_t info;
        void __user *pc;

        pc = (void __user *)instruction_pointer(regs);

        if (processor_mode(regs) == SVC_MODE) {
#ifdef CONFIG_THUMB2_KERNEL
                if (thumb_mode(regs)) {
                        instr = __mem_to_opcode_thumb16(((u16 *)pc)[0]);
                        if (is_wide_instruction(instr)) {
                                u16 inst2;
                                inst2 = __mem_to_opcode_thumb16(((u16 *)pc)[1]);
                                instr = __opcode_thumb32_compose(instr, inst2);
                        }
                } else
#endif
                        instr = __mem_to_opcode_arm(*(u32 *) pc);
        } else if (thumb_mode(regs)) {
                if (get_user(instr, (u16 __user *)pc))
                        goto die_sig;
                instr = __mem_to_opcode_thumb16(instr);
                if (is_wide_instruction(instr)) {
                        unsigned int instr2;
                        if (get_user(instr2, (u16 __user *)pc+1))
                                goto die_sig;
                        instr2 = __mem_to_opcode_thumb16(instr2);
                        instr = __opcode_thumb32_compose(instr, instr2);
                }
        } else {
                if (get_user(instr, (u32 __user *)pc))
                        goto die_sig;
                instr = __mem_to_opcode_arm(instr);
        }

        if (call_undef_hook(regs, instr) == 0)
                return;

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;
```

```
54.             info.si_addr  = pc;
55.
56.             arm_notify_die("Oops - undefined instruction", regs, &info, 0, 6);
57.     }
```

只有user mode application produce underfined instruction exception才会到die_sig。

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

在打开UDBG_UNDEFINED的情况下，会打印出那个application引起的该exception，以及出错的instruction address.

2. invalid system call

```
1.  static int bad_syscall(int n, struct pt_regs *regs)
2.  {
3.          struct thread_info *thread = current_thread_info();
4.          siginfo_t info;
5.
6.          if ((current->personality & PER_MASK) != PER_LINUX &&
7.              thread->exec_domain->handler) {
8.                  thread->exec_domain->handler(n, regs);
9.                  return regs->ARM_r0;
10.         }
11.
12. #ifdef CONFIG_DEBUG_USER
13.         if (user_debug & UDBG_SYSCALL) {
14.                 printk(KERN_ERR "[%d] %s: obsolete system call %08x.\n",
15.                         task_pid_nr(current), current->comm, n);
16.                 dump_instr(KERN_ERR, regs);
17.         }
18. #endif
19.
20.         info.si_signo = SIGILL;
21.         info.si_errno = 0;
22.         info.si_code  = ILL_ILLTRP;
23.         info.si_addr  = (void __user *)instruction_pointer(regs) -
24.                          (thumb_mode(regs) ? 2 : 4);
25.
26.         arm_notify_die("Oops - bad syscall", regs, &info, n, 0);
27.
28.         return regs->ARM_r0;
29. }
```

3. data abort

```
/*
 * 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);
}
```

in arch/arm/mm/fault.c

```
/*
 * Something tried to access memory that isn't in our memory map..
 * User mode accesses just cause a SIGSEGV
 */
static void
__do_user_fault(struct task_struct *tsk, unsigned long addr,
                unsigned int fsr, unsigned int sig, int code,
                struct pt_regs *regs)
{
        struct siginfo si;

#ifdef CONFIG_DEBUG_USER
        if (((user_debug & UDBG_SEGV) && (sig == SIGSEGV)) ||
            ((user_debug & UDBG_BUS)  && (sig == SIGBUS))) {
                printk(KERN_DEBUG "%s: unhandled page fault (%d) at 0x%08lx, code 0x%03x\n",
                       tsk->comm, sig, addr, fsr);
                show_pte(tsk->mm, addr);
                show_regs(regs);
        }
#endif

        tsk->thread.address = addr;
        tsk->thread.error_code = fsr;
        tsk->thread.trap_no = 14;
        si.si_signo = sig;
        si.si_errno = 0;
        si.si_code = code;
        si.si_addr = (void __user *)addr;
        force_sig_info(sig, &si, tsk);
}
```

application access invalid memory, kernel could dump application name!

网上google到一个利用"user_debug"feature定位bug的case。

运行busbox,有时候正常，有时候报"illegal instruction"

查错如下：

1. build kernel, make CONFIG_DEBUG_USER=y

user_debug=1

2. run busybox

3. 当执行mdev时出错，从log可看到pc=000ca8b4,即000ca8b4处是invalid instruction.

4. objdump -d busybox, disassemble busybox

5.在000ca8b4处发觉是clz instruction。

clz 指令体系结构：

此 ARM 指令可用于 ARMv5 及更高版本。

此 32 位 Thumb 指令可用于 ARMv6T2 及更高版本。

此指令无 16 位 Thumb 版本

6. modify compile option

CFLAGS_busybox += -static -march=**armv4t**