pkmap_countabs() and abs64()

```
/*
 * abs() handles unsigned and signed longs, ints, shorts and chars.  For all
 * input types abs() returns a signed long.
 * abs() should not be used for 64-bit types (s64, u64, long long) - use abs64()
 * for those.
 */
#define abs(x) ({                                               \
                long ret;                                       \
                if (sizeof(x) == sizeof(long)) {                \是否可抢占 （preemptable）
                        long __x = (x);                         \
                        ret = (__x < 0) ? -__x : __x;           \
                } else {                                        \
                        int __x = (x);                          \
                        ret = (__x < 0) ? -__x : __x;           \
                }                                               \
                ret;                                            \
        })

#define abs64(x) ({                             \
                s64 __x = (x);                  \
                (__x < 0) ? -__x : __x;         \
        })
```

判断是否可抢占（ preemptable ）

每个process的thread_info都有一个preempt_count。

```
struct thread_info {
        unsigned long           flags;          /* low level flags */
int                     preempt_count;  /* 0 => preemptable, <0 => bug */
        mm_segment_t            addr_limit;     /* address limit */
        struct task_struct      *task;          /* main task structure */
        struct exec_domain      *exec_domain;   /* execution domain */
        __u32                   cpu;            /* cpu */
        __u32                   cpu_domain;     /* cpu domain */
        struct cpu_context_save cpu_context;    /* cpu context */
        __u32                   syscall;        /* syscall number */
        __u8                    used_cp[16];    /* thread used copro */
        unsigned long           tp_value[2];    /* TLS registers */
#ifdef CONFIG_CRUNCH
        struct crunch_state     crunchstate;
#endif
        union fp_state          fpstate __attribute__((aligned(8)));
        union vfp_state         vfpstate;
#ifdef CONFIG_ARM_THUMBEE
        unsigned long           thumbee_state;  /* ThumbEE Handler Base register
*/
#endif
        struct restart_block    restart_block;
};
```

```
#define preempt_enable() \
do { \
        barrier(); \
        if (unlikely(preempt_count_dec_and_test())) \
                __preempt_schedule(); \
} while (0)
```

preempt_count_dec_and_test（）check preempt_count - 1后是否为0，若是则

```
asmlinkage __visible void __sched notrace preempt_schedule(void)
{
        /*
         * If there is a non-zero preempt_count or interrupts are disabled,
         * we do not want to preempt the current task. Just return..
         */
        if (likely(!preemptible()))
                return;

        do {
                __preempt_count_add(PREEMPT_ACTIVE);
                __schedule();
                __preempt_count_sub(PREEMPT_ACTIVE);

                /*
                 * Check again in case we missed a preemption opportunity
                 * between schedule and now.
                 */
                barrier();
        } while (need_resched());
}
NOKPROBE_SYMBOL(preempt_schedule);
```

define preemptible() (preempt_count() == 0 && !irqs_disabled())

可抢占的判断标准

1. 当前运行process的preempt_count为0（原来为1，减去1后）
2. diable irq

preempt_count值的含义比较复杂

```
/*
 * We put the hardirq and softirq counter into the preemption
 * counter. The bitmask has the following meaning:
 *
 * - bits 0-7 are the preemption count (max preemption depth: 256)
 * - bits 8-15 are the softirq count (max # of softirqs: 256)
 *
 * The hardirq count could in theory be the same as the number of
 * interrupts in the system, but we run all interrupt handlers with
 * interrupts disabled, so we cannot have nesting interrupts. Though
 * there are a few palaeontologic drivers which reenable interrupts in
 * the handler, so we need more than one bit here.
 *
 * PREEMPT_MASK:        0x000000ff
 * SOFTIRQ_MASK:        0x0000ff00
 * HARDIRQ_MASK:        0x000f0000
 *     NMI_MASK:        0x00100000
 * PREEMPT_ACTIVE:      0x00200000
 */
#define PREEMPT_BITS    8
#define SOFTIRQ_BITS    8
#define HARDIRQ_BITS    4
#define NMI_BITS        1
```

即只有bit0 - bit7是对preempt的counter，bit8 - bit15 is for softirq

bit16-bit19 is for hardirq, bit20 is for NMI

临时禁止preemptable

因为现在kernel也是部分可抢占的，所以如果kernel中有code希望临时禁止preemptable，可如下

preempt_disable();

do something

preempt_enable();

check kernel是否运行于“特殊”状态

```
#define in_irq()                (hardirq_count())
#define in_softirq()            (softirq_count())
#define in_interrupt()          (irq_count())
#define in_serving_softirq()    (softirq_count() & SOFTIRQ_OFFSET)
```

in_irq() --- hardware interrupt running 判断

```
#define irq_count()     (preempt_count() & (HARDIRQ_MASK | SOFTIRQ_MASK \
                                | NMI_MASK))
```

check preempt_count的bit16 - bit19是否为0来判断是否运行在hardware interrupt中。

in_softirq() --- software irq running 判断

in_interrupt()

```
#define irq_count()     (preempt_count() & (HARDIRQ_MASK | SOFTIRQ_MASK \
                                | NMI_MASK))
```

"interrupt"是 **hardware interrupt, software interrupt and NMI** running的合称。

check 当前是否可抢占

```
#ifdef CONFIG_PREEMPT_COUNT
# define preemptible()  (preempt_count() == 0 && !irqs_disabled())
#else
# define preemptible()  0
#endif
```

enable / disable pagefault

include/linux/uaccess.h

```
/*
 * These routines enable/disable the pagefault handler in that
 * it will not take any locks and go straight to the fixup table.
 *
 * They have great resemblance to the preempt_disable/enable calls
 * and in fact they are identical; this is because currently there is
 * no other way to make the pagefault handlers do this. So we do
 * disable preemption but we don't necessarily care about that.
 */
static inline void pagefault_disable(void)
{
        preempt_count_inc();
        /*
         * make sure to have issued the store before a pagefault
         * can hit.
         */
        barrier();
}

static inline void pagefault_enable(void)
{
#ifndef CONFIG_PREEMPT
        /*
         * make sure to issue those last loads/stores before enabling
         * the pagefault handler again.
         */
        barrier();
        preempt_count_dec();
#else
        preempt_enable();
#endif
}
```

从code上看，更准确的function name应该是current_process_pagefault_disable()

该函数只是disable当前process产生pagefault。因为当产生pagefault时必然伴随着current process要抢占(它要被schedule出去sleep).

但问题是，pagefault产生了，但又不能被抢占,那current process该怎么办？有点糊涂了？！

要看page fault handler才能明白怎么办。

access kernel memory safely

mm/mmaccess.c

```
/**
 * probe_kernel_read(): safely attempt to read from a location
 * @dst: pointer to the buffer that shall take the data
 * @src: address to read from
 * @size: size of the data chunk
 *
 * Safely read from address @src to the buffer at @dst.  If a kernel fault
 * happens, handle that and return -EFAULT.
 */

long __weak probe_kernel_read(void *dst, const void *src, size_t size)
    __attribute__((alias("__probe_kernel_read")));

long __probe_kernel_read(void *dst, const void *src, size_t size)
{
        long ret;
        mm_segment_t old_fs = get_fs();

        set_fs(KERNEL_DS);
pagefault_disable();
        ret = __copy_from_user_inatomic(dst,
                        (__force const void __user *)src, size);
        pagefault_enable();
        set_fs(old_fs);

        return ret ? -EFAULT : 0;
}
EXPORT_SYMBOL_GPL(probe_kernel_read);

/**
 * probe_kernel_write(): safely attempt to write to a location
 * @dst: address to write to
 * @src: pointer to the data that shall be written
 * @size: size of the data chunk
 *
 * Safely write to address @dst from the buffer at @src.  If a kernel fault
 * happens, handle that and return -EFAULT.
 */
long __weak probe_kernel_write(void *dst, const void *src, size_t size)
    __attribute__((alias("__probe_kernel_write")));

long __probe_kernel_write(void *dst, const void *src, size_t size)
{
        long ret;
        mm_segment_t old_fs = get_fs();

        set_fs(KERNEL_DS);
        pagefault_disable();
        ret = __copy_to_user_inatomic((__force void __user *)dst, src, size);
        pagefault_enable();
        set_fs(old_fs);

        return ret ? -EFAULT : 0;
```

```
}
EXPORT_SYMBOL_GPL(probe_kernel_write);
```

分析如下：

pagefault_disable();

ret = __copy_from_user_inatomic(dst,

(__force const void __user *)src, size);

在kernel产生page fault，进入do_page)fault() / arch/arm/mm/fault.c

```
static int __kprobes
do_page_fault(unsigned long addr, unsigned int fsr, struct pt_regs *regs)
{
        struct task_struct *tsk;
        struct mm_struct *mm;
        int fault, sig, code;
        unsigned int flags = FAULT_FLAG_ALLOW_RETRY | FAULT_FLAG_KILLABLE;

        if (notify_page_fault(regs, fsr))
                return 0;

        tsk = current;
        mm  = tsk->mm;

        /* Enable interrupts if they were enabled in the parent context. */
        if (interrupts_enabled(regs))
                local_irq_enable();

        /*
         * If we're in an interrupt or have no user
         * context, we must not take the fault..
         */
        if (in_atomic() || !mm)
                goto no_context;

        if (user_mode(regs))
                flags |= FAULT_FLAG_USER;
        if (fsr & FSR_WRITE)
                flags |= FAULT_FLAG_WRITE;

......
```

运行到

if (in_atomic() || !mm)

goto no_context;

由于pagefault_disable()的缘故，这里的in_atomic()返回true，执行流运行如下code

```
no_context:
        __do_kernel_fault(mm, addr, fsr, regs);
        return 0;
```

```
/*
 * Oops.  The kernel tried to access some page that wasn't present.
 */
static void
__do_kernel_fault(struct mm_struct *mm, unsigned long addr, unsigned int fsr,
                  struct pt_regs *regs)
{
        /*
         * Are we prepared to handle this kernel fault?
         */
        if (fixup_exception(regs))
                return;

        /*
         * No handler, we'll have to terminate things with extreme prejudice.
         */
        bust_spinlocks(1);
        printk(KERN_ALERT
                "Unable to handle kernel %s at virtual address %08lx\n",
                (addr < PAGE_SIZE) ? "NULL pointer dereference" :
                "paging request", addr);

        show_pte(mm, addr);
        die("Oops", regs, fsr);
        bust_spinlocks(0);
        do_exit(SIGKILL);
}
```

由于

__copy_from_user_inatomic(dst, (__force const void __user *)src, size)会安装exception handler，所以

这里的fixup_exception(regs))会返回true。

```
int fixup_exception(struct pt_regs *regs)
{
        const struct exception_table_entry *fixup;

        fixup = search_exception_tables(instruction_pointer(regs));
        if (fixup) {
                regs->ARM_pc = fixup->fixup;
#ifdef CONFIG_THUMB2_KERNEL
                /* Clear the IT state to avoid nasty surprises in the fixup */
                regs->ARM_cpsr &= ~PSR_IT_MASK;
#endif
        }

        return fixup != NULL;
}
```