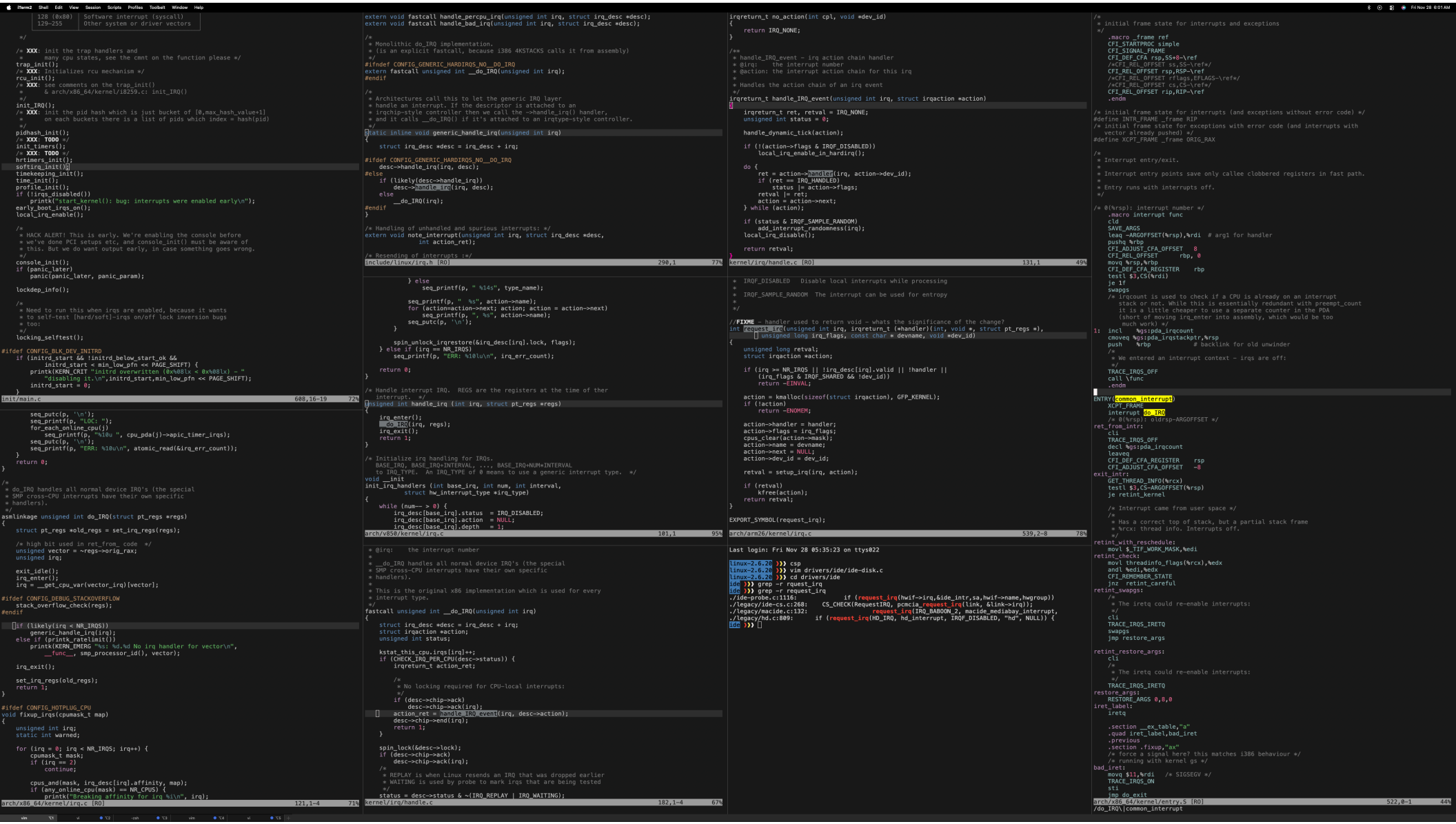
iTerm2 Shell Edit View Session Scripts Profiles Toolbelt Window Help
Fri Nov 28 8:01 AM
128 (0x80) Software interrupt (syscall)
129–255 Other system or driver vectors
*/
/* XXX: init the trap handlers and
* many cpu states, see the cmnt on the function please */
trap_init();
/* XXX: initializes rcu mechanism */
rcu_init();
/* XXX: see comments on the trap_init()
* & arch/x86_64/kernel/i8259.c: init_IRQ()
*/
init_IRQ();
/* XXX: init the pid hash which is just bucket of [0,max_hash_value+1]
* on each buckets there is a list of pids which index = hash(pid)
*/
pidhash_init();
/* XXX: TODO */
init_timers();
/* XXX: TODO */
hrtimers_init();
softirq_init();
timekeeping_init();
time_init();
profile_init();
if (!irqs_disabled())
printk("start_kernel(): bug: interrupts were enabled early\n");
early_boot_irqs_on();
local_irq_enable();
/*
* HACK ALERT! This is early. We're enabling the console before
* we've done PCI setups etc, and console_init() must be aware of
* this. But we do want output early, in case something goes wrong.
*/
console_init();
if (panic_later)
panic(panic_later, panic_param);
lockdep_info();
/*
* Need to run this when irqs are enabled, because it wants
* to self-test [hard/soft]-irqs on/off lock inversion bugs
* too:
*/
locking_selftest();
#ifdef CONFIG_BLK_DEV_INITRD
if (initrd_start && !initrd_below_start_ok &&
initrd_start < min_low_pfn << PAGE_SHIFT) {
printk(KERN_CRIT "initrd overwritten (0x%08lx < 0x%08lx) - "
"disabling it.\n",initrd_start,min_low_pfn << PAGE_SHIFT);
initrd_start = 0;
}
init/main.c 608,16-19 72%
seq_putc(p, '\n');
seq_printf(p, "LOC: ");
for_each_online_cpu(j)
seq_printf(p, "%10u ", cpu_pda(j)->apic_timer_irqs);
seq_putc(p, '\n');
seq_printf(p, "ERR: %10u\n", atomic_read(&irq_err_count));
}
return 0;
}
/*
* do_IRQ handles all normal device IRQ's (the special
* SMP cross-CPU interrupts have their own specific
* handlers).
*/
asmlinkage unsigned int do_IRQ(struct pt_regs *regs)
{
struct pt_regs *old_regs = set_irq_regs(regs);
/* high bit used in ret_from_ code */
unsigned vector = ~regs->orig_rax;
unsigned irq;
exit_idle();
irq_enter();
irq = __get_cpu_var(vector_irq)[vector];
#ifdef CONFIG_DEBUG_STACKOVERFLOW
stack_overflow_check(regs);
#endif
if (likely(irq < NR_IRQS))
generic_handle_irq(irq);
else if (printk_ratelimit())
printk(KERN_EMERG "%s: %d.%d No irq handler for vector\n",
__func__, smp_processor_id(), vector);
irq_exit();
set_irq_regs(old_regs);
return 1;
}
#ifdef CONFIG_HOTPLUG_CPU
void fixup_irqs(cpumask_t map)
{
unsigned int irq;
static int warned;
for (irq = 0; irq < NR_IRQS; irq++) {
cpumask_t mask;
if (irq == 2)
continue;
cpus_and(mask, irq_desc[irq].affinity, map);
if (any_online_cpu(mask) == NR_CPUS) {
printk("Breaking affinity for irq %i\n", irq);
arch/x86_64/kernel/irq.c [RO] 121,1–4 71%
extern void fastcall handle_percpu_irq(unsigned int irq, struct irq_desc *desc);
extern void fastcall handle_bad_irq(unsigned int irq, struct irq_desc *desc);
/*
* Monolithic do_IRQ implementation.
* (is an explicit fastcall, because i386 4KSTACKS calls it from assembly)
*/
#ifndef CONFIG_GENERIC_HARDIRQS_NO__DO_IRQ
extern fastcall unsigned int __do_IRQ(unsigned int irq);
#endif
/*
* Architectures call this to let the generic IRQ layer
* handle an interrupt. If the descriptor is attached to an
* irqchip-style controller then we call the ->handle_irq() handler,
* and it calls __do_IRQ() if it's attached to an irqtype-style controller.
*/
static inline void generic_handle_irq(unsigned int irq)
{
struct irq_desc *desc = irq_desc + irq;
#ifdef CONFIG_GENERIC_HARDIRQS_NO__DO_IRQ
desc->handle_irq(irq, desc);
#else
if (likely(desc->handle_irq))
desc->handle_irq(irq, desc);
else
__do_IRQ(irq);
#endif
}
/* Handling of unhandled and spurious interrupts: */
extern void note_interrupt(unsigned int irq, struct irq_desc *desc,
int action_ret);
/* Resending of interrupts :*/
include/linux/irq.h [RO] 290,1 77%
seq_printf(p, " %14s", type_name);
seq_printf(p, " %s", action->name);
for (action=action->next; action; action = action->next)
seq_printf(p, ", %s", action->name);
seq_putc(p, '\n');
}
spin_unlock_irqrestore(&irq_desc[irq].lock, flags);
} else if (irq == NR_IRQS)
seq_printf(p, "ERR: %10u\n", irq_err_count);
return 0;
}
/* Handle interrupt IRQ. REGS are the registers at the time of ther
interrupt. */
unsigned int handle_irq (int irq, struct pt_regs *regs)
{
irq_enter();
do_IRQ(irq, regs);
irq_exit();
return 1;
}
/* Initialize irq handling for IRQs.
BASE_IRQ, BASE_IRQ+INTERVAL, ... BASE_IRQ+NUM*INTERVAL
to IRQ_TYPE. An IRQ_TYPE of 0 means to use a generic interrupt type. */
void __init
init_irq_handlers (int base_irq, int num, int interval,
struct hw_interrupt_type *irq_type)
{
while (num-- > 0) {
irq_desc[base_irq].status = IRQ_DISABLED;
irq_desc[base_irq].action = NULL;
irq_desc[base_irq].depth = 1;
arch/v850/kernel/irq.c 101,1 95%
* @irq: the interrupt number
*
* __do_IRQ handles all normal device IRQ's (the special
* SMP cross-CPU interrupts have their own specific
* handlers).
*
* This is the original x86 implementation which is used for every
* interrupt type.
*/
fastcall unsigned int __do_IRQ(unsigned int irq)
{
struct irq_desc *desc = irq_desc + irq;
struct irqaction *action;
unsigned int status;
kstat_this_cpu.irqs[irq]++;
if (CHECK_IRQ_PER_CPU(desc->status)) {
irqreturn_t action_ret;
/*
* No locking required for CPU-local interrupts:
*/
if (desc->chip->ack)
desc->chip->ack(irq);
action_ret = handle_IRQ_event(irq, desc->action);
desc->chip->end(irq);
return 1;
}
spin_lock(&desc->lock);
if (desc->chip->ack)
desc->chip->ack(irq);
/*
* REPLAY is when Linux resends an IRQ that was dropped earlier
* WAITING is used by probe to mark irqs that are being tested
*/
status = desc->status & ~(IRQ_REPLAY | IRQ_WAITING);
kernel/irq/handle.c 182,1–4 67%
irqreturn_t no_action(int cpl, void *dev_id)
{
return IRQ_NONE;
}
/**
* handle_IRQ_event - irq action chain handler
* @irq: the interrupt number
* @action: the interrupt action chain for this irq
*
* Handles the action chain of an irq event
*/
irqreturn_t handle_IRQ_event(unsigned int irq, struct irqaction *action)
{
irqreturn_t ret, retval = IRQ_NONE;
unsigned int status = 0;
handle_dynamic_tick(action);
if (!(action->flags & IRQF_DISABLED))
local_irq_enable_in_hardirq();
do {
ret = action->handler(irq, action->dev_id);
if (ret == IRQ_HANDLED)
status |= action->flags;
retval |= ret;
action = action->next;
} while (action);
if (status & IRQF_SAMPLE_RANDOM)
add_interrupt_randomness(irq);
local_irq_disable();
return retval;
}
kernel/irq/handle.c [RO] 131,1 49%
* IRQF_DISABLED - Disable local interrupts while processing
*
* IRQF_SAMPLE_RANDOM The interrupt can be used for entropy
*
*/
//FIXME - handler used to return void - whats the significance of the change?
int request_irq(unsigned int irq, irqreturn_t (*handler)(int, void *, struct pt_regs *),
unsigned long irq_flags, const char * devname, void *dev_id)
{
unsigned long retval;
struct irqaction *action;
if (irq >= NR_IRQS || !irq_desc[irq].valid || !handler ||
(irq_flags & IRQF_SHARED && !dev_id))
return -EINVAL;
action = kmalloc(sizeof(struct irqaction), GFP_KERNEL);
if (!action)
return -ENOMEM;
action->handler = handler;
action->flags = irq_flags;
cpus_clear(action->mask);
action->name = devname;
action->next = NULL;
action->dev_id = dev_id;
retval = setup_irq(irq, action);
if (retval)
kfree(action);
return retval;
}
EXPORT_SYMBOL(request_irq);
arch/arm26/kernel/irq.c 539,2–8 78%
Last login: Fri Nov 28 05:35:23 on ttys022
linux-2.6.20 ❯❯❯ csp
linux-2.6.20 ❯❯❯ vim drivers/ide/ide-disk.c
linux-2.6.20 ❯❯❯ cd drivers/ide
ide ❯❯❯ grep -r rquest_irq
ide ❯❯❯ grep -r request_irq
./ide-probe.c:1116: if (request_irq(hwif->irq,&ide_intr,sa,hwif->name,hwgroup))
./legacy/ide-cs.c:268: CS_CHECK(RequestIRQ, pcmcia_request_irq(link, &link->irq));
./legacy/macide.c:132: request_irq(IRQ_BABOON_2, macide_mediabay_interrupt,
./legacy/hd.c:809: if (request_irq(HD_IRQ, hd_interrupt, IRQF_DISABLED, "hd", NULL)) {
ide ❯❯❯
/*
* initial frame state for interrupts and exceptions
*/
.macro _frame ref
CFI_STARTPROC simple
CFI_SIGNAL_FRAME
CFI_DEF_CFA rsp,SS+8-\ref
/*CFI_REL_OFFSET ss,SS-\ref*/
CFI_REL_OFFSET rsp,RSP-\ref
/*CFI_REL_OFFSET rflags,EFLAGS-\ref*/
/*CFI_REL_OFFSET cs,CS-\ref*/
CFI_REL_OFFSET rip,RIP-\ref
.endm
/* initial frame state for interrupts (and exceptions without error code) */
#define INTR_FRAME _frame RIP
/* initial frame state for exceptions with error code (and interrupts with
vector already pushed) */
#define XCPT_FRAME _frame ORIG_RAX
/*
* Interrupt entry/exit.
*
* Interrupt entry points save only callee clobbered registers in fast path.
*
* Entry runs with interrupts off.
*/
/* 0(%rsp): interrupt number */
.macro interrupt func
cld
SAVE_ARGS
leaq -ARGOFFSET(%rsp),%rdi # arg1 for handler
pushq %rbp
CFI_ADJUST_CFA_OFFSET 8
CFI_REL_OFFSET rbp, 0
movq %rsp,%rbp
CFI_DEF_CFA_REGISTER rbp
testl $3,CS(%rdi)
je 1f
swapgs
/* irqcount is used to check if a CPU is already on an interrupt
stack or not. While this is essentially redundant with preempt_count
it is a little cheaper to use a separate counter in the PDA
(short of moving irq_enter into assembly, which would be too
much work) */
1: incl %gs:pda_irqcount
cmoveq %gs:pda_irqstackptr,%rsp
push %rbp # backlink for old unwinder
/*
* We entered an interrupt context - irqs are off:
*/
TRACE_IRQS_OFF
call \func
.endm
ENTRY(common_interrupt)
XCPT_FRAME
interrupt do_IRQ
/* 0(%rsp): oldrsp-ARGOFFSET */
ret_from_intr:
cli
TRACE_IRQS_OFF
decl %gs:pda_irqcount
leaveq
CFI_DEF_CFA_REGISTER rsp
CFI_ADJUST_CFA_OFFSET -8
exit_intr:
GET_THREAD_INFO(%rcx)
testl $3,CS-ARGOFFSET(%rsp)
je retint_kernel
/* Interrupt came from user space */
/*
* Has a correct top of stack, but a partial stack frame
* %rcx: thread info. Interrupts off.
*/
retint_with_reschedule:
movl $_TIF_WORK_MASK,%edi
retint_check:
movl threadinfo_flags(%rcx),%edx
andl %edi,%edx
CFI_REMEMBER_STATE
jnz retint_careful
retint_swapgs:
/*
* The iretq could re-enable interrupts:
*/
cli
TRACE_IRQS_IRETQ
swapgs
jmp restore_args
retint_restore_args:
cli
/*
* The iretq could re-enable interrupts:
*/
TRACE_IRQS_IRETQ
restore_args:
RESTORE_ARGS 0,8,0
iret_label:
iretq
.section __ex_table,"a"
.quad iret_label,bad_iret
.previous
.section .fixup,"ax"
/* force a signal here? this matches i386 behaviour */
/* running with kernel gs */
bad_iret:
movq $11,%rdi /* SIGSEGV */
TRACE_IRQS_ON
sti
jmp do_exit
arch/x86_64/kernel/entry.S [RO] 522,0-1 44%
/do_IRQ\|common_interrupt