Simple Introduction to paravirt_ops for Xen

Zhiqiang Ma

We use the function raw_local_irq_disable() and raw_local_irq_enable() functions in Linux kernel to introduce paravirt_ops for Xen and Xenified kernel.

Xenified Linux kernel

```
--- linux-2.6.32.13.orig/arch/x86/include/mach-xen/asm/irqflags.h
+++ linux-2.6.32.13/arch/x86/include/mach-xen/asm/irqflags.h

+#define raw_local_irq_disable()        xen_irq_disable()
+
+#define raw_local_irq_enable() xen_irq_enable()


+#define xen_irq_disable()                                              \
+do {                                                                   \
+       vcpu_info_write(evtchn_upcall_mask, 1);                         \
+       barrier();                                                      \
+} while (0)
+
+#define xen_irq_enable()                                               \
+do {                                                                   \
+       vcpu_info_t *_vcpu;                                             \
+       barrier();                                                      \
+       _vcpu = current_vcpu_info();                                    \
+       _vcpu->evtchn_upcall_mask = 0;                                  \
+       barrier(); /* unmask then check (avoid races) */               \
+       if (unlikely(_vcpu->evtchn_upcall_pending))                     \
+               force_evtchn_callback();                                \
+} while (0)
```

paravirt_ops implementation in Linux

```
--- a/include/asm-i386/paravirt.h
+++ b/include/asm-i386/paravirt.h

+struct paravirt_ops
+{
+       unsigned int kernel_rpl;
+
+       /* All the function pointers here are declared as "fastcall"
+          so that we get a specific register-based calling
+          convention.  This makes it easier to implement inline
+          assembler replacements. */
+
+       void (fastcall *cpuid)(unsigned int *eax, unsigned int *ebx,
+                              unsigned int *ecx, unsigned int *edx);
+
+       unsigned int (fastcall *get_debugreg)(int regno);
+       void (fastcall *set_debugreg)(int regno, unsigned int value);
+
+       void (fastcall *sync_core)(void);
+
```

```
+	void (fastcall *clts)(void);
+
+	unsigned int (fastcall *read_cr0)(void);
+	void (fastcall *write_cr0)(unsigned int);
+
+	unsigned int (fastcall *read_cr2)(void);
+	void (fastcall *write_cr2)(unsigned int);
+
+	unsigned int (fastcall *read_cr3)(void);
+	void (fastcall *write_cr3)(unsigned int);
+
+	unsigned int (fastcall *read_cr4_safe)(void);
+	unsigned int (fastcall *read_cr4)(void);
+	void (fastcall *write_cr4)(unsigned int);
+
+	unsigned long (fastcall *save_fl)(void);
+	void (fastcall *restore_fl)(unsigned long);
+	unsigned long (fastcall *save_fl_irq_disable)(void);
+	void (fastcall *irq_disable)(void);
+	void (fastcall *irq_enable)(void);
+	void (fastcall *safe_halt)(void);
+	void (fastcall *halt)(void);
+	void (fastcall *wbinvd)(void);
+
+	/* err = 0/-EFAULT.  wrmsr returns 0/-EFAULT. */
+	u64 (fastcall *read_msr)(unsigned int msr, int *err);
+	int (fastcall *write_msr)(unsigned int msr, u64 val);
+
+	u64 (fastcall *read_tsc)(void);
+	u64 (fastcall *read_pmc)(void);
+
+	void (fastcall *load_tr_desc)(void);
+	void (fastcall *load_ldt_desc)(void);
+	void (fastcall *load_gdt)(const struct Xgt_desc_struct *);
+	void (fastcall *load_idt)(const struct Xgt_desc_struct *);
+	void (fastcall *store_gdt)(struct Xgt_desc_struct *);
+	void (fastcall *store_idt)(struct Xgt_desc_struct *);
+	unsigned long (fastcall *store_tr)(void);
+	void (fastcall *load_tls)(struct thread_struct *t, unsigned int cpu);
+	void (fastcall *write_ldt_entry)(void *dt, int entrynum, u64 entry);
+	void (fastcall *write_gdt_entry)(void *dt, int entrynum, u64 entry);
+	void (fastcall *write_idt_entry)(void *dt, int entrynum, u64 entry);
+
+	void (fastcall *set_iopl_mask)(unsigned mask);
+
+	/* These two are jmp to, not actually called. */
+	void (fastcall *irq_enable_sysexit)(void);
+	void (fastcall *iret)(void);
+};
+
+extern struct paravirt_ops paravirt_ops;

+static inline void raw_local_irq_disable(void)
+{
+	paravirt_ops.irq_disable();
+}
+
+static inline void raw_local_irq_enable(void)
+{
+	paravirt_ops.irq_enable();
+}
```

```
--- /dev/null
+++ b/arch/i386/kernel/paravirt.c

+static fastcall void nopara_irq_disable(void)
+{
+       __asm__ __volatile__("cli": : :"memory");
+}
+
+static fastcall void nopara_irq_enable(void)
+{
+       __asm__ __volatile__("sti": : :"memory");
+}
+

+struct paravirt_ops paravirt_ops = {
+       .kernel_rpl = 0,
+       .cpuid = nopara_cpuid,
+       .get_debugreg = nopara_get_debugreg,
+       .set_debugreg = nopara_set_debugreg,
+       .sync_core = nopara_sync_core,
+       .clts = nopara_clts,
+       .read_cr0 = nopara_read_cr0,
+       .write_cr0 = nopara_write_cr0,
+       .read_cr2 = nopara_read_cr2,
+       .write_cr2 = nopara_write_cr2,
+       .read_cr3 = nopara_read_cr3,
+       .write_cr3 = nopara_write_cr3,
+       .read_cr4 = nopara_read_cr4,
+       .read_cr4_safe = nopara_read_cr4_safe,
+       .write_cr4 = nopara_write_cr4,
+       .save_fl = nopara_save_fl,
+       .restore_fl = nopara_restore_fl,
+       .save_fl_irq_disable = nopara_save_fl_irq_disable,
+       .irq_disable = nopara_irq_disable,
+       .irq_enable = nopara_irq_enable,
+       .safe_halt = nopara_safe_halt,
+       .halt = nopara_halt,
+       .wbinvd = nopara_wbinvd,
+       .read_msr = nopara_read_msr,
+       .write_msr = nopara_write_msr,
+       .read_tsc = nopara_read_tsc,
+       .read_pmc = nopara_read_pmc,
+       .load_tr_desc = nopara_load_tr_desc,
+       .load_ldt_desc = nopara_load_ldt_desc,
+       .load_gdt = nopara_load_gdt,
+       .load_idt = nopara_load_idt,
+       .store_gdt = nopara_store_gdt,
+       .store_idt = nopara_store_idt,
+       .store_tr = nopara_store_tr,
+       .load_tls = nopara_load_tls,
+       .write_ldt_entry = nopara_write_ldt_entry,
+       .write_gdt_entry = nopara_write_gdt_entry,
+       .write_idt_entry = nopara_write_idt_entry,
+
+       .set_iopl_mask = nopara_set_iopl_mask,
+       .irq_enable_sysexit = nopara_irq_enable_sysexit,
+       .iret = nopara_iret,
+};
```

```
+EXPORT_SYMBOL_GPL(paravirt_ops);
```

arch/x86/xen/irq.c

```
static void xen_irq_disable(void)
{
        /* There's a one instruction preempt window here.  We need to
           make sure we're don't switch CPUs between getting the vcpu
           pointer and updating the mask. */
        preempt_disable();
        percpu_read(xen_vcpu)->evtchn_upcall_mask = 1;
        preempt_enable_no_resched();
}
PV_CALLEE_SAVE_REGS_THUNK(xen_irq_disable);

static void xen_irq_enable(void)
{
        struct vcpu_info *vcpu;

        /* We don't need to worry about being preempted here, since
           either a) interrupts are disabled, so no preemption, or b)
           the caller is confused and is trying to re-enable interrupts
           on an indeterminate processor. */

        vcpu = percpu_read(xen_vcpu);
        vcpu->evtchn_upcall_mask = 0;

        /* Doesn't matter if we get preempted here, because any
           pending event will get dealt with anyway. */

        barrier(); /* unmask then check (avoid races) */
        if (unlikely(vcpu->evtchn_upcall_pending))
                xen_force_evtchn_callback();
}
PV_CALLEE_SAVE_REGS_THUNK(xen_irq_enable);
```