CS 444/544 OS II
Lab Tutorial #9
Preemptive Multitasking,
and Inter-process Communication
(Lab4 – Part C)

Acknowledgement: Slides drawn heavily from Yeongjin Jiang
Exercise 13/14: Enable Timer-interrupt-based Preemptive Multitasking

• We will now enable timer-based preemptive multitasking, the mechanism that we learn in the lecture

• To do this, you need to do the following:
  • 1) write TRAPHANDLER / IDT entry to Hardware IRQs
  • 2) handle timer interrupt
  • 3) enable interrupt in user mode (ring 3)
  • 4) enable interrupt in the scheduler (ring 0)
Exercise 13/14: Enable Timer-interrupt-based Preemptive Multitasking

• 1) write TRAPHANDLER / IDT entry to Hardware IRQs

```assembly
TRAPHANDLER_NOEC(t_irq_timer, IRQ_OFFSET + IRQ_TIMER);  // 32 + 0
TRAPHANDLER_NOEC(t_irq_kbd, IRQ_OFFSET + IRQ_KBD);      // 32 + 1
TRAPHANDLER_NOEC(t_irq_2, IRQ_OFFSET + 2);              // 32 + 2
TRAPHANDLER_NOEC(t_irq_3, IRQ_OFFSET + 3);              // 32 + 3
TRAPHANDLER_NOEC(t_irq_serial, IRQ_OFFSET + IRQ_SERIAL);  // 32 + 4
TRAPHANDLER_NOEC(t_irq_5, IRQ_OFFSET + 5);              // 32 + 5
TRAPHANDLER_NOEC(t_irq_6, IRQ_OFFSET + 6);              // 32 + 6
TRAPHANDLER_NOEC(t_irq_spurious, IRQ_OFFSET + IRQ_SPURIOUS);  // 32 + 7
TRAPHANDLER_NOEC(t_irq_8, IRQ_OFFSET + 8);              // 32 + 8
TRAPHANDLER_NOEC(t_irq_9, IRQ_OFFSET + 9);              // 32 + 9
TRAPHANDLER_NOEC(t_irq_10, IRQ_OFFSET + 10);            // 32 + 10
TRAPHANDLER_NOEC(t_irq_11, IRQ_OFFSET + 11);            // 32 + 11
TRAPHANDLER_NOEC(t_irq_12, IRQ_OFFSET + 12);            // 32 + 12
TRAPHANDLER_NOEC(t_irq_13, IRQ_OFFSET + 13);            // 32 + 13
TRAPHANDLER_NOEC(t_irq_ide, IRQ_OFFSET + IRQ_IDE);      // 32 + 14
TRAPHANDLER_NOEC(t_irq_15, IRQ_OFFSET + 15);            // 32 + 15
```
Exercise 13/14: Enable Timer-interrupt-based Preemptive Multitasking

• 1) write TRAPHANDLER / IDT entry to Hardware IRQs

```c
SETGATE(idt[IRQ_OFFSET + IRQ_TIMER], 0, GD_KT, t_irq_timer, 0);
SETGATE(idt[IRQ_OFFSET + IRQ_KBD], 0, GD_KT, t_irq_kbd, 0);
SETGATE(idt[IRQ_OFFSET + 2], 0, GD_KT, t_irq_2, 0);
SETGATE(idt[IRQ_OFFSET + 3], 0, GD_KT, t_irq_3, 0);
SETGATE(idt[IRQ_OFFSET + IRQ_SERIAL], 0, GD_KT, t_irq_serial, 0);
SETGATE(idt[IRQ_OFFSET + 5], 0, GD_KT, t_irq_5, 0);
SETGATE(idt[IRQ_OFFSET + 6], 0, GD_KT, t_irq_6, 0);
SETGATE(idt[IRQ_OFFSET + IRQ_SPURIOUS], 0, GD_KT, t_irq_spurious, 0);
SETGATE(idt[IRQ_OFFSET + 8], 0, GD_KT, t_irq_8, 0);
SETGATE(idt[IRQ_OFFSET + 9], 0, GD_KT, t_irq_9, 0);
SETGATE(idt[IRQ_OFFSET + 10], 0, GD_KT, t_irq_10, 0);
SETGATE(idt[IRQ_OFFSET + 11], 0, GD_KT, t_irq_11, 0);
SETGATE(idt[IRQ_OFFSET + 12], 0, GD_KT, t_irq_12, 0);
SETGATE(idt[IRQ_OFFSET + 13], 0, GD_KT, t_irq_13, 0);
SETGATE(idt[IRQ_OFFSET + IRQ_IDE], 0, GD_KT, t_irq_ide, 0);
SETGATE(idt[IRQ_OFFSET + 15], 0, GD_KT, t_irq_15, 0);
```
Exercise 13/14: Enable Timer-interrupt-based Preemptive Multitasking

• 2) handle timer interrupt

• In trap_dispatch()

```c
case (IRQ_OFFSET + IRQ_TIMER):
{
    lapic_eoi();
    sched_yield();
}
```

• Meaning
  • If timer interrupt arrives, we schedule another process to support preemptive multitasking!
Exercise 13/14: Enable Timer-interrupt-based Preemptive Multitasking

• 3) enable interrupt in user mode (ring 3)

• In env_alloc() in kern/env.c

```c
// Enable interrupts while in user mode.
// LAB 4: Your code here.
e->env_tf.tf_eflags |= FL_IF;
```

• This will enable receiving interrupt during user execution
Exercise 13/14: Enable Timer-interrupt-based Preemptive Multitasking

• 4) enable interrupt in the scheduler (ring 0)

• In sched_halt() in kern/sched.c

```c
// Reset stack pointer, enable interrupts and then halt.
asm volatile ( 
   "movl $0, %ebp\n"
   "movl %0, %esp\n"
   "pushl $0\n"
   "pushl $0\n"
   "pushl $0\n"
   // LAB 4:
   // Uncomment the following line after completing exercise 13
   "sti\n"
   "1:\n"
   "hlt\n"
   "jmp 1b\n"
   : : "a" (thiscpu->cpu_ts.tsEsp0));
```
Now You Should Get ALL OKs up to SPIN

• Check TRAPHANDLER, IDT, trap_dispatch, or enabling/disabling interrupt if your JOS does not switch among environment correctly...
Caveat

- Kernel Panic: interrupt is not disabled

```
kernel panic on CPU 0 at kern/trap.c:414: assertion failed: !(read_eflags() & FL_IF)
```

- If you get this error, this could be happening if

  ```c
  SETGATE(idt[T_SYSCALL], 1, GD_KT, t_syscall, 3);
  ```

- You set the 2\textsuperscript{nd} arg of SETGATE as 1

- This flag is for enabling/disabling interrupt while handling another interrupt
  - So we must set it as 0 for all SETGATE for the current JOS implementation
Exercise 15: Implementing IPC

• Inter-process Communication (IPC)
  • A communication channel between two processes (environments)

• Process does not share memory space
  • The same virtual address will be backed by different physical pages

• Then, how can we send a message?

![Diagram of two processes: ENV 1 (Sender) and ENV 2 (Receiver), with an arrow pointing from Sender to Receiver labeled "Send “1234”!"]
Exercise 15: send/recv via Kernel

• How kernel mediates message passing between 2 envs?

• Receiver (*sys_ipc_recv*)
  • Indicate the env is waiting for a message
    • env_ipc_recving = 1
  • Because it must wait until recv the msg,
    • Set env_status = NOT_RUNNABLE
    • DO NOT RUN this if it waits for IPC msg
  • Set tf_regsregs_eax = 0
    • It will return 0 if recv succeeds
  • Run sched_yield()
    • sys_ipc_recv will never directly return 0
    • env_pop_ret will return 0 from tf..
Exercise 15: send/recv via Kernel

• How kernel mediates message passing between 2 envs?

• Sender (sys_ipc_try_send)
  • Check if target envid is waiting for IPC
    • if (e2->env_ipc_recving == 1)
  • Send the value via env_ipc_value
    • e2->env_ipc_value = 1234;
  • Set who sent the value
    • e2->env_ipc_from = curenv->env_id
  • Set e2->env_status as
    • ENV_RUNNABLE
Exercise 15: send/recv via Kernel

• How kernel mediates message passing between 2 envs?

After ENV1 sets ENV2’s status as **ENV_RUNNABLE**, then ENV2 can be scheduled and run

sys_ipc_recv returns via env_pop_tf

How can we get the value 1234?

**thisenv->env_ipc_value**

How can we get who sent the value?

**thisenv->env_ipc_from**

Send “1234”!

**sys_ipc_recv()**

**sys_ipc_try_send()**

Sender

Receiver

JOS KERNEL

struct Env *e1

struct Env *e2

env_ipc_recving = 1

env_status = **ENV_RUNNABLE**

tf_regs.regs_eax = 0

env_ipc_value = 1234

env_ipc_from
Exercise 15: How to Send a Page?

• Now we know how to send a 4 byte data (value)
  • Store that in env’s env_ipc_value

• Can we send more than 4 bytes (e.g., sending 512 bytes at once) ?
  • 1. Use value to indicate the size of data (e.g., 512 bytes)
  • 2. Put a 512-byte data in a physical page (from sender)
  • 3. Sender maps the page at dstva of Receiver ENV
  • 4. After receiver gets the value (from env_ipc_value == 512)
    • Read that amount of data from dstva
Exercise 15: Send a Page

Send “512 byte of data”!

Map the page for srcva to dstva!
Exercise 15: Some hints

• Use page_lookup and page_insert to
  • Get the PTE of srcva
  • Get the corresponding physical page of srcva (struct PageInfo *pp)
  • Put pp to dstva via page_insert
  • Also set e->env_ipc_perm (get the perm from the PTE of srcva)
Exercise 15: Some hints

• In lib/ipc.c
  • `sys_ipc_recv` never returns if there is no error
    • It will internally run `sched_yield()` -> then `env_run()` will schedule it back
    • So pass the return value via `tf_regs.regs_eax = 0`

  • `ipc_send` must wait if receiving env is not ready
    • `sys_ipc_try_end` returns –E_IPC_NOT_RECV
    • Then stay in a while loop and keep try to send...

• NULL is not an invalid address for `srcva/dstva`
  • Put higher address than UTOP, e.g., KERNBASE?
Exercise 15: Some hints

• When submitting lab4, make your JOS run fast enough to pass grading script

• DO NOT use too many cprintf
  • Primes could be VERY SLOW
  • Removing debug printing will let you finish this within 30 seconds...