K-OS Task Management

Kai Chen

CS134c Spring 2003

Overview

- **Notion of process**
  - **Task State Segment**
  - **Context Switch**
- **Scheduling**
  - **Simple round-robin**
- **Process synchronization**
  - **Semaphores (todo)**

Task State Segment

```
unsigned short back_link, reserved0;
unsigned long esp0;
unsigned short ss0, reserved1;
unsigned long esp1;
unsigned short ss1, reserved2;
unsigned long esp2;
unsigned short ss2, reserved3;
unsigned long cr3, eip, eflags;
unsigned long eax, ecx, edx, ebx, esp, ebp, esi, edi;
unsigned short es, reserved4;
unsigned short cs, reserved5;
unsigned short ss, reserved6;
unsigned short ds, reserved7;
unsigned short fs, reserved8;
unsigned short gs, reserved9;
unsigned short ldt, reserved10;
unsigned short tflag, iomap;
```

Task State Segment (Continued)

- Reflect the initial state of the registers
- At the very least:
  - CR3: currently set to kernel page directory base, until we create a new address space for each process
  - EIP: set to the entry point of the task
  - Segment register: proper values
    - CS: CODE selector
    - DS, ES, FS, GS, SS … : DATA selector
  - ESP: Top of the stack set aside for the task

TSS as GDT entry

- A TSS needs to be added to GDT before any context switch.
- A TSS selector has:
  - base address
  - limit: size-1 = 103 bytes
  - busy TSS: type 11
  - available TSS: type 9
  - protection, etc ...

- GDT Layout updated:
  - NULL selector
  - CODE selector
  - DATA selector
  - TSS0 selector
  - TSS1 selector
  - TSS2 selector

  ... ...

Context Switch

- **TSS contains the proper values**
- **TSS is installed in GDT**
- **Far jump to a TSS selector to perform context switch:**
  - **TSS0 has selector 0x18**
  - **TSS1 has selector 0x20**
  - **etc ...**

Scheduling (with Round-Robin)

- A timeout counter for preemption.
- Timer interrupt handler decreases the counter. If it reaches zero, call the schedule() function.
- Scheduler (schedule()) perform context switch to the next task in the runqueue.

Task Structure

```
struct task_t {
        tss_t tss;
        unsigned short tss_sel;
        enum{
                TASK_RUNNING = 0,
                TASK_INTERRUPTIBLE = 1,
                TASK_UNINTERRUPTIBLE = 2,
                TASK_NON_EXIST = 3
        } status;
        struct task_t *next;  // for runqueue, wait_queue …
}
```

Current State

- **We have:**
  - **TSS0 is set up. So init_task is running.**
  - **Tasks are statically allocated (16 max). TSS' are inserted into the GDT.**
  - **Routines to create tasks and perform context switches.**
  - **A simple scheduler.**
  - **Timer interrupt handler.**
- **Problems:**
  - **General protection fault (INT 13) or CPU triple fault on context switches.**
  - **Probably overlooked some technical details ...**