COP4610: Operating Systems

Xv6 Scheduling

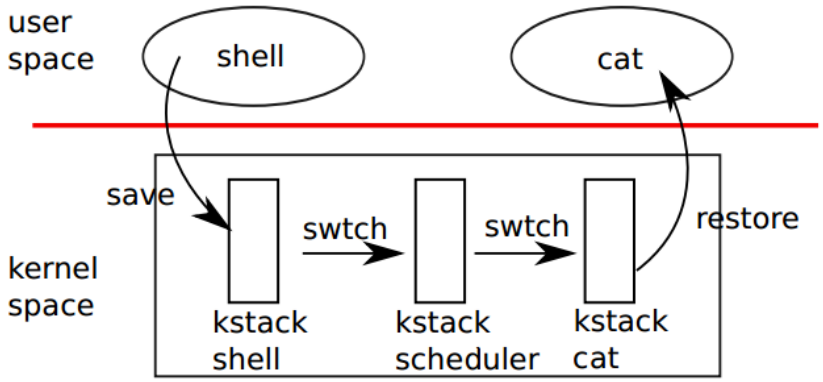
Zhi Wang

Florida State University

Spring 2015

Xv6 Scheduling

- Xv6 supports preemptive scheduling
  - process waiting for I/O, or for child to exit, or waiting in `sleep`
  - a timer periodically forces a context switch (freq = 100 ticks/s)
- Xv6 implements a round-robin scheduler



- Xv6 performs context switch in two steps
  - ➠ a process's kernel thread to the current CPU's scheduler thread
  - ➠ the scheduler thread picks the next process for execution, and switches to the selected process
- Context switch is implemented in swtch.s
  - ➠ `void swtch(struct context **old, struct context *new);`

Struct Conext (proc.h)

```
// Saved registers for kernel context switches. Don't need to save all the
// segment registers (%cs, etc), because they are constant across kernel
// contexts.  Don't need to save %eax, %ecx, %edx, because the x86
// convention is that the caller has saved them. Contexts are stored at
// the bottom of the stack they describe; the stack pointer is the address
// of the context. The layout of the context matches the layout of the
// stack in swtch.S at the "Switch stacks" comment. Switch doesn't save
// eip explicitly, but it is on the stack and allocproc() manipulates it.
struct context {
    uint   edi;
    uint   esi;
    uint   ebx;
    uint   ebp;
    uint   eip;
};
```

- Struct context normally sits at the top of the stack
  - ➠ the parameters of swtch point to the top of the stack

Swtch (swtch.s)

```
.globl swtch
swtch:
movl 4(%esp), %eax
movl 8(%esp), %edx

# Save old callee-save registers
pushl %ebp
pushl %ebx
pushl %esi
pushl %edi

# Switch stacks
movl %esp, (%eax)
movl %edx, %esp

# Load new callee-save registers
popl %edi
popl %esi
popl %ebx
popl %ebp
ret
```

Scheduler

- Each CPU has its own scheduler kernel thread (scheduler)
  - ➠ each CPU executes scheduler after initialization (in mpmain)
- Context switches in Xv6 are performed in two steps
  - ➠ current process →scheduler (in sched function)
  - ➠ scheduler →the next process (in schduler function)

Scheduler (Continued)

- Scheduler loops through the ptable to select the next process
  ➠ a round-robin scheduling algorithm
- It calls swtch to switch to the kernel thread of the next process
  ➠ kernel stack and the registers are saved and restored
  ➠ swtch returns when the running process calls sched to switch to the scheduler
- A process can call yield to voluntarily give up CPU
  ➠ a running process yields to other processes in the timer interrupt handler for preemptive scheduling (trap.c)

Project 2: Priority-based Schduler for Xv6

- Goal: to implement a priority-based scheduler for Xv6
- Steps:
  - ➠ add a system call to set the priority of a process (see project 1)
  - ➠ change the scheduler function to select the process with the highest priority
- Requirements:
  - ➠ correctly use the ptable lock (follow the current scheduler)
  - ➠ "if there are multiple processes with the same highest priority, the scheduler uses round-robin to execute them in turn to avoid starvation."