236376 Operating Systems Engineering
Recitation #4:
Processes and switching

Presented by: Igor Smolyar
<igors@cs>
Author: Muli Ben-Yehuda
<muli@cs>
Plan

- Processes
- Threads
- Scheduling
Abstract virtual machine

Provides the illusion to application of a dedicated computer, but an abstract one convenient for application developer

One process cannot affect another accidentally
Process API

- fork
- exec
- exit
- wait
- kill
- sbrk
- getpid
Problem: more processes than processors

- xv6 picture: 1 user thread and 1 kernel thread per process; 1 scheduler thread per processor; n processors
- A process: address space plus one or more threads
- A thread: thread of execution
- Kernel thread: thread running in kernel mode
- User thread: thread running in user mode
- Thread of execution: an abstraction that contains enough state of a running program that it can be stopped and resumed
- xv6 API: yield, swtch
Goals for solution

- Switching transparent to user threads
- User thread cannot hog a processor (kernel thread assumed to be correct, so not a goal)
Overview of switch between two user threads

- **user threads:**
  - User -> kernel transition
  - kernel -> kernel switch
  - kernel -> User transition

- **guaranteed U->K transitions**
  - timing interrupt every 100 ms
  - switches to different kernel thread on yield
  - the different kernel thread returns to a different user thread
Challenges in implementing:

- Opaque code ("You are not supposed to understand this")
- Concurrency (several processors switching between threads)
- Terminating a thread, always need a valid stack
xv6 design:

- One scheduler thread per processor
- Scheduling organized as co-routines
- Scheduler thread performs cleanup
- proc.c:scheduler()
- swtch.S:swtch()
- trap.c:trap()
- proc.c:yield()