CS3210: Coordination

Taesoo Kim

Administrivia

- Lab5 out, due April 11
- Demo day: 8-min for demo, 2-min for Q&A
- Final project (write-up): April 22

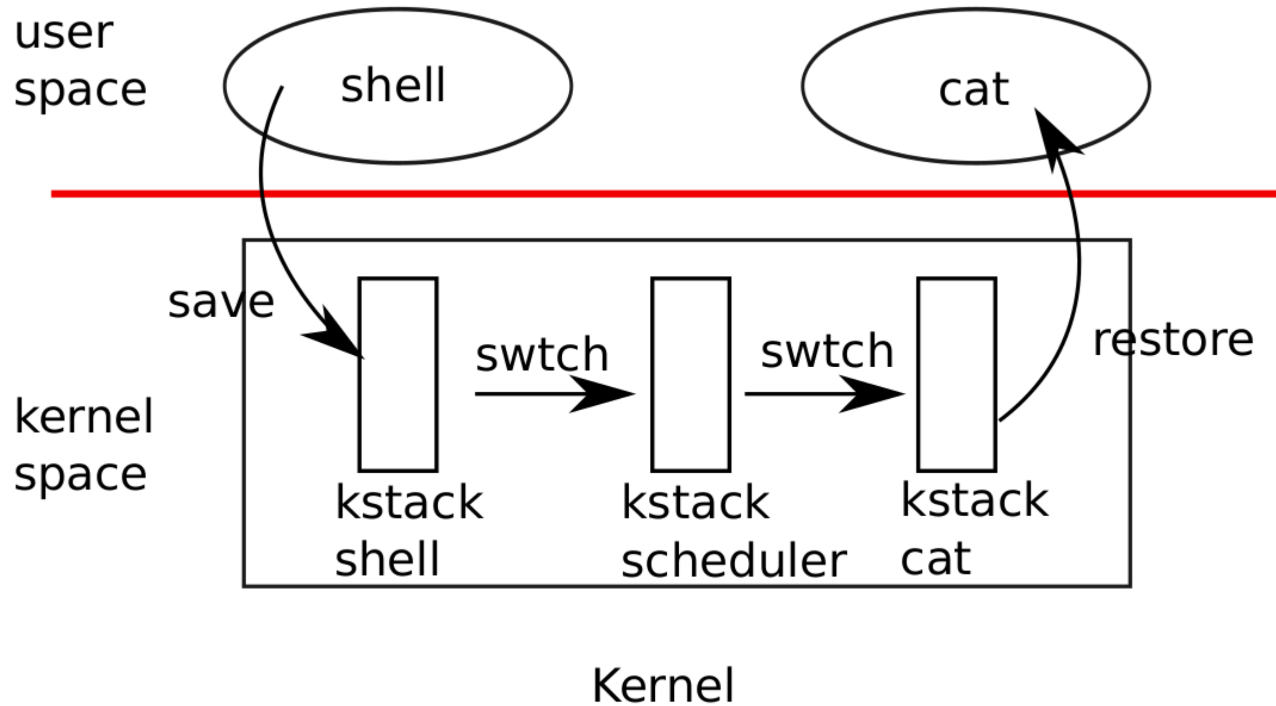
Summary of last lectures

- Power-on → BIOS → bootloader → kernel → user programs
- OS: abstraction, multiplexing, isolation, sharing
- Design: monolithic (xv6) vs. micro kernels (jos)
- Abstraction: process, system calls
- Isolation mechanisms: CPL, segmentation, paging

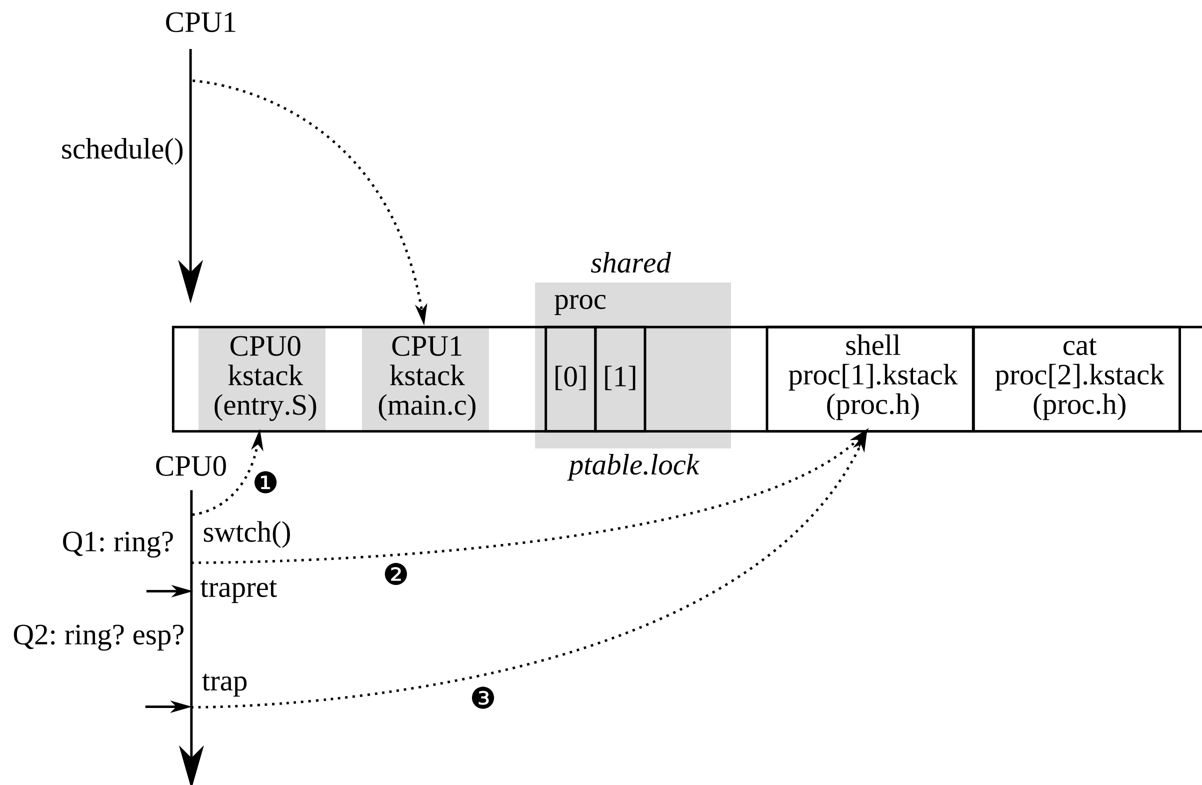
Today's plan

- Context switching (i.e., `swtch` and `sched`) in detail
- Sequence coordination
    - xv6: sleep & wakeup
- Challenges
    - Lost wakeup problem
    - Signals

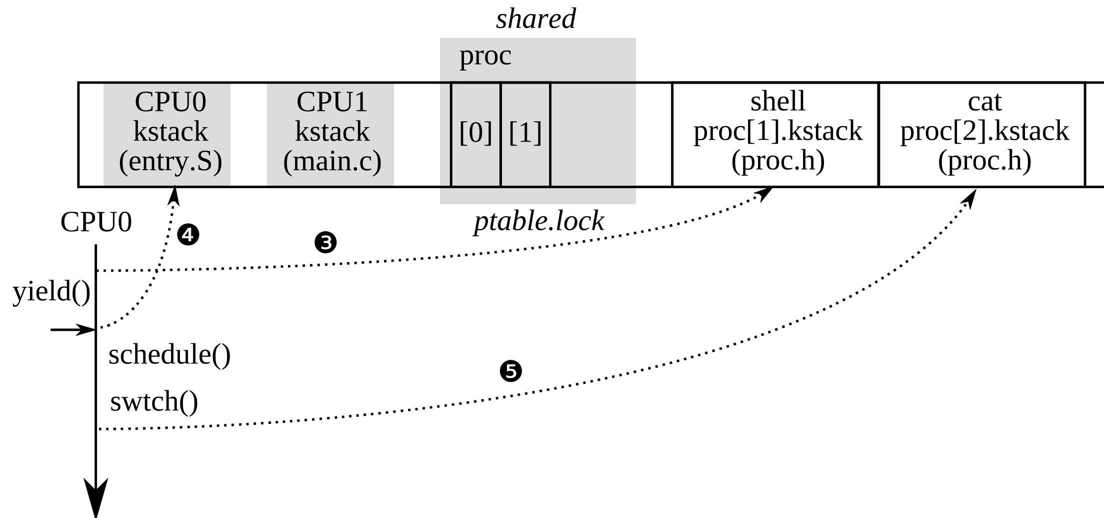
Big picture: switching



Switching overview: CPU perspective



Switching overview: CPU perspective



Code

- `swtch()`, `scheduler()`, `sched()`
- about `ptable.lock`

DEMO: sched

```
br sched
commands
p cpus[cpunum()].proc.pid
c
end
```

Big picture

- Multiple threads executing in the kernel
- Sharing memory, devices, and various data structures.
- Locks to protect invariants
- One outstanding disk request
- One scheduler selecting a thread to run

Sequence coordination

- How to arrange for threads to wait for each other to do
  - e.g., wait for disk interrupt to complete
  - e.g., wait for pipe readers to make space in pipe
  - e.g., wait for child to exit
  - e.g., wait for block to use

Strawman solution: spin

```
01    struct q { void *ptr; };
02
03    void* send(struct q *q, void *p) {
04      while(q->ptr != 0)
05        ;
06      q->ptr = p;
07    }
08
09    void* recv(struct q *q) {
10      void *p;
11      while((p = q->ptr) == 0)
12        ;
13      q->ptr = 0;
14      return p;
15    }
```

Strawman solution: spin

- Q: cpu0 send(), cpu1 recv()?
- Q: cpu0 send(), cpu1 send()?
- Q: cpu0 recv(), cpu1 send()?
- Q: cpu0 recv(), cpu1 recv()?
- Q: problem?

Better solution: primitives for coordination

- Sleep & wakeup (xv6)
- Condition variables (e.g., `pthread_cond`)
- Barriers (next tutorial)

Sleep & wakeup

- `sleep(chan)`
  - sleeps on a "channel", an address to name the condition we are sleeping on
- `wakeup(chan)`
  - wakeup wakes up all threads sleeping on `chan`
  - this may wake up more than one thread

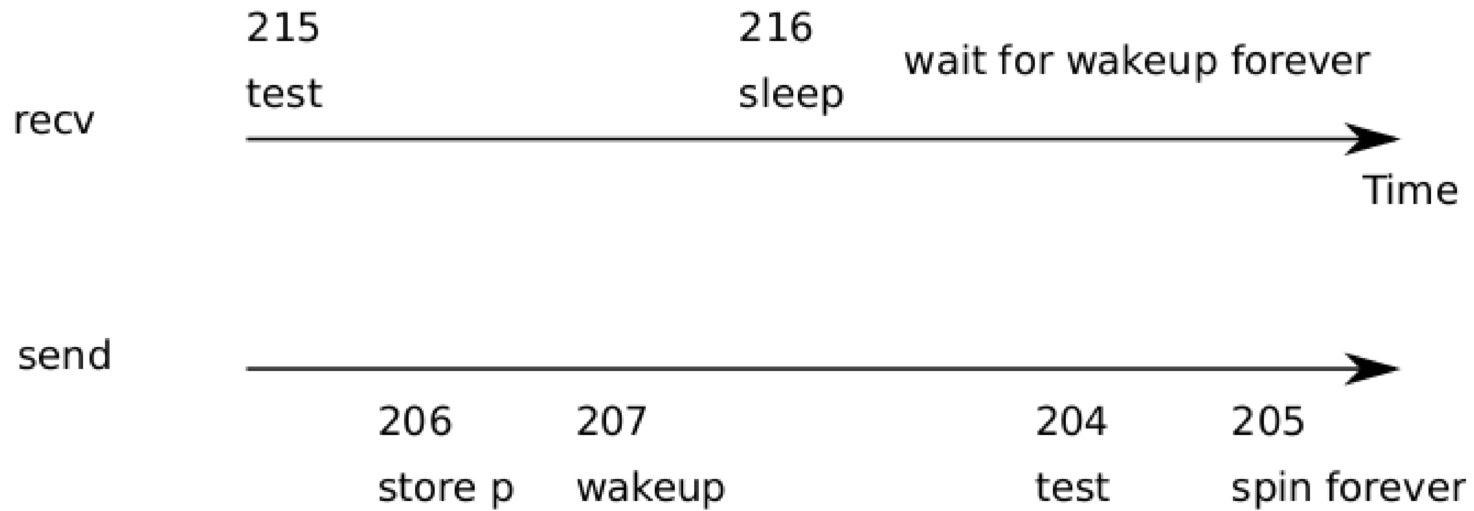
Attempt 1: sleep & wakeup

```
01    void* send(struct q *q, void *p) {
02      while(q->ptr != 0)
03        ;
04      q->ptr = p;
05      wakeup(q);  /* Q? */
06    }
07
08    void* recv(struct q *q) {
09      void *p;
10      while((p = q->ptr) == 0)
11        sleep(q);  /* Q? */
12      q->ptr = 0;
13      return p;
14    }
```

Strawman solution: spin

- Q: cpu0 send(), cpu1 recv()?
- Q: cpu0 send(), cpu1 send()?
- Q: cpu0 recv(), cpu1 send()?
- Q: cpu0 recv(), cpu1 recv()?
- Q: problem? (hint: concurrently run `while` in `send/recv`)

Lost wakeup problem



Attempt1: fixing the lost wakeup problem

- Q: how to atomically run the code (checking and sleeping)?

```
10        while((p = q->ptr) == 0)
11          sleep(q);
```

Attempt1: fixing the lost wakeup problem

- Let's use a spinlock

```
struct q {
  struct spinlock lock;
  void *ptr;
};
```

Attempt1: fixing the lost wakeup problem

```
01    void* send(struct q *q, void *p) {
02      acquire(&q->lock);
03      while(q->ptr != 0)
04        ;
05      q->ptr = p;
06      wakeup(q);
07      release(&q->lock);
08    }
09
10    void* recv(struct q *q) {
11      void *p;
12      acquire(&q->lock);
13      while((p = q->ptr) == 0)
14        sleep(q);
15      q->ptr = 0;
```

Problems?

- Q: cpu0 send(), cpu1 recv()?
- Q: cpu0 send(), cpu1 send()?
- Q: cpu0 recv(), cpu1 send()?
- Q: cpu0 recv(), cpu1 recv()?

Attempt2: releasing the lock when sleeping

```
01    void* send(struct q *q, void *p) {
02      acquire(&q->lock);
03      while(q->ptr != 0)
04        ;
05      q->ptr = p;
06      wakeup(q);
07      release(&q->lock);
08    }
09
10    void* recv(struct q *q) {
11      void *p;
12      acquire(&q->lock);
13      while((p = q->ptr) == 0)
14        sleep(q, &q->lock);
15      q->ptr = 0;
```

Problems?

- Q: cpu0 send(), cpu1 recv()?
- Q: cpu0 send(), cpu1 send()?
- Q: cpu0 recv(), cpu1 send()?
- Q: cpu0 recv(), cpu1 recv()?
- We need a similar treatment for `send()` (i.e., `sleep()`)

Code

- `sleep()`, `wakeup()`
- about: `ptable.lock`

Summary: sleep takes a lock as argument

- Sleeper and wakeup acquires locks for shared data structure
- `sleep()` holds the lock until after it has `ptable.lock`
- Once it has `ptable.lock`, no wakeup can come in before it sets state to sleeping → no lost wakeup problem
- Requires that sleep takes a lock argument!

Case study: ide (blockio)

- Device I/O is too slow to just spin (wait) for its competition
- `bread(b)` → `iderw(b)`
  - it waits (sleep) until the requests block is ready
- `trap()` → `ideintr()`
  - it notifies (wakeup) the waiter

Example: iderw

- code: `iderw()` (sleeper), `ideintr()` (wakeup)
- Q: wakeup cannot get lock until sleeper is already to at sleep, why a loop around sleep?

```
01        // Wait for request to finish.
02        while((b&#8594;flags & (B_VALID|B_DIRTY)) != B_VALID){
03          sleep(b, &idelock);
04        }
```

Another example: pipe

- What is the race if sleep didn't take p→lock as argument?

Many primitives in literature to solve lost-wakeup problem

- Counting wakeup&sleep calls in semaphores
- Pass locks as an extra argument in condition variables (as in sleep)

References

- Intel Manual
- UW CSE 451
- OSPP
- MIT 6.828
- Wikipedia
- The Internet