Today: Synchronization Wrap-up

• Abstraction
  – (mutex) Locks
  – Condition variables
  – Semaphores – Today!

• Pitfalls
  – Deadlock – Today!
Synchronization: Abstractions

- Mutex locks
- Condition variables
- Semaphores
Synchronization: Locks

- `pthread_mutex_t`
- `pthread_mutex_{try}lock`
- `pthread_mutex_unlock`

Natural fit for
- Enforcing exclusive access to critical section

Pitfalls
- Lock too much/little
- Forget to unlock after locking
- **Deadlock**, livelock, etc…
Synchronization: CVs

• Condition *variable*?
  – Do they really maintain any state that the user cares about?
• `pthread_cond_t`
• `pthread_cond_{timed}wait`
• `pthread_cond_{signal,broadcast}`
Synchronization: Semaphores (1/4)

- **State**
  - count, queue
  - count > 0 → queue is empty

- **Up / Signal / V / Unlock / Post**
  - Unblock a waiter, or if none, then increase count

- **Down / Wait / P / Lock**
  - Decrease the count, or if already 0, add the caller to wait queue
Synchronization: Semaphores (2/4)

- semaphore.h
- sem_t
- sem_init
- sem_{try, timed}wait
Synchronization: Semaphores (3/4)

• Implement semaphore using locks?

```python
wait():
    keep_waiting = true
    while keep_waiting:
        lock.acquire()
        if count > 0
            count--
            keep_waiting = false
        lock.release()
```

• What would signal do?
• How can we avoid this busy-waiting?
Synchronization: Semaphores (4/4)

- Implement semaphore using locks... and CVs?

```python
wait():
    lock.acquire()
    while count <= 0:
        cond.wait(lock)
    count--
    lock.release()
```

- Why `while`, and not `if`?

Deadlock: Trivial Example

- Thread A
  - lock(x)
  - lock(y)

- Thread B
  - lock(y)
  - lock(x)

- Problem?
Deadlock: Conditions

- Mutual exclusion
- Hold-and-wait
- No preemption
- Circular wait

Let’s philosophize.
Deadlock: Solutions

• Bury your head in the sand?
• Order locks
  – What condition does this avoid?
  – philosophize_ordered.c
• Trylock
  – philosophize_trylock.c
• Other ideas?
Questions?