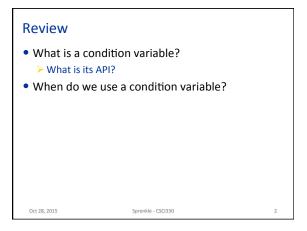
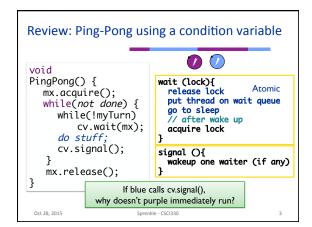
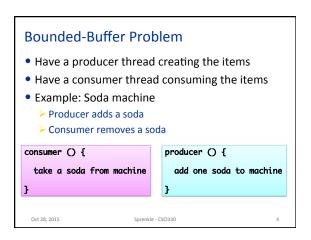
Today • Concurrency Problems ➤ Producer Consumer • Bounded Buffer • Pipes ➤ Dining Philosophers • Synchronization Mechanisms ➤ Condition Variables







```
Solving producer-consumer

• What are the variables/shared state?

> Soda machine buffer

> Number of sodas in machine (≤ MaxSodas)

• Locks?

> 1 to protect all shared state (sodaLock)

• Mutual exclusion?

> Only one thread can manipulate machine at a time

• Ordering constraints?

> Consumer must wait if machine is empty (CV hasSoda)

> Producer must wait if machine is full (CV hasRoom)

Oct 28, 2015

Sprenkle - CSCI330

5
```

```
Producer-consumer code

consumer () {
lock
wait if empty
take a soda from machine
notify (not full)
unlock
}

oct 28, 2015

producer () {
lock
wait if full
add one soda to machine
notify (not empty)
unlock
}
```

```
Producer-consumer code
consumer () {
                             producer () {
                               sodaLock.acquire()
  sodaLock.acquire()
  while (numSodas == 0) {
                               while(numSodas==MaxSodas){
    hasSoda.wait(sodaLock)
                                 hasRoom.wait(sodaLock)
  take a soda from machine
                               add one soda to machine
 hasRoom.signal()
                               hasSoda.signal()
  sodaLock.release()
                               sodaLock.release()
}
 Oct 28, 2015
                        Sprenkle - CSCI330
```

```
>1 Resource, >1 Consumers
   The signal should be a broadcast
   if the producer can produce more than one resource,
   and there are multiple consumers.
consumer () {
                            producer () {
  sodaLock.acquire()
                              sodaLock.acquire()
 while (numSodas == 0) {
                              while(numSodas==MaxSodas){
   hasSoda.wait(sodaLock)
                                hasRoom.wait(sodaLock)
  take a soda from machine
                              fill machine with soda
 signal(hasRoom)
                              broadcast(hasSoda)
 sodaLock.release()
                              sodaLock.release()
```

```
Can I always use broadcast instead of signal?
Yes, assuming threads recheck condition
And they should: "loop before you leap"!
Another thread could get to the lock before wait returns
Why might I use signal instead?
Efficiency -- May wakeup threads for no good reason
```

Sprenkle - CSCI330

Oct 28, 2015

```
Condition Variable Design Pattern
methodThatWaits() {
                             methodThatSignals() {
   lock.acquire();
                                lock.acquire();
   // Read/write shared
                                // Read/write shared
   // state
                                // state
                                // If testSharedState is
   while (
     testSharedState()) {
                                // now true
      cv.wait(lock):
                                cv.signal(lock);
   // Read/write shared
                                // Read/write shared
                                // state
   // state
                                lock.release();
   lock.release();
 Oct 28, 2015
                        Sprenkle - CSCI330
                                                      10
```

```
Summary: Condition Variables

• Condition variable is memoryless

• If signal when no one is waiting, no op

• Wait atomically releases lock

• What if wait, then release?

• What if release, then wait?

wait (lock){
    release lock
    put thread on wait queue
    go to sleep
    // after wake up
    acquire lock

Oct 28, 2015
```

Summary: Condition Variables When a thread is woken up from wait, it may not run immediately Signal/broadcast puts thread on ready (not running) list When lock is released, anyone might acquire it Benefit: simplifies implementation Of condition variables and locks Of code that uses condition variables and locks

Sprenkle - CSCI330

Oct 28, 2015

Using Condition Variables

- Document the condition(s) associated with each
 - What are the waiters waiting for?
 - > When can a waiter expect a signal?
- ALWAYS hold lock when calling wait, signal, broadcast
 - > Condition variable is sync FOR shared state
 - > ALWAYS hold lock when accessing shared state

Oct 28, 2015

Sprenkle - CSCI330

1330

Using Condition Variables

 Wait MUST be in a loop -- "Loop before you leap!" while (needToWait()) { condition.wait(lock);

}

- > Another thread may beat you to the mutex.
- > The signaler may be careless.
 - Some thread packages have "spurious wakeups":
 threads woken up, though a single signal has taken place
- > A single CV may have multiple conditions
- Signals on CVs do not stack!
 - A signal will be lost if nobody is waiting: always check the wait condition before calling wait.

Oct 28, 2015

Sprenkle - CSCI330

Looking Ahead

- More synchronization mechanisms
 - > Semaphores
- More synchronization problems
 - Dining Philosophers
- Still working through Chapter 5 of text book
- Project 3: next Wednesday

Oct 28, 2015

Sprenkle - CSCI330