

CS162 Operating Systems and Systems Programming Lecture 5

Semaphores, Conditional Variables

February 1, 2012
Anthony D. Joseph and Ion Stoica
<http://inst.eecs.berkeley.edu/~cs162>

Goals for Today

- Continue with Synchronization Abstractions
 - Semaphores, Monitors and condition variables
- Readers-Writers problem and solution

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne. Many slides generated from lecture notes by Kubiatiowicz.

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.2

Where are we going with synchronization?

Programs	Shared Programs
Higher-level API	Locks Semaphores Monitors Send/Receive
Hardware	Load/Store Disable Ints Test&Set Comp&Swap

- We are going to implement various higher-level synchronization primitives using atomic operations
 - Everything is pretty painful if only atomic primitives are load and store
 - Need to provide primitives useful at user-level

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.3

Semaphores



- Semaphores are a kind of generalized locks
 - First defined by Dijkstra in late 60s
 - Main synchronization primitive used in original UNIX
- Definition: a Semaphore has a non-negative integer value and supports the following two operations:
 - **P()**: an atomic operation that waits for semaphore to become positive, then decrements it by 1
 - » Think of this as the wait() operation
 - **V()**: an atomic operation that increments the semaphore by 1, waking up a waiting P, if any
 - » Think of this as the signal() operation
 - Note that **P()** stands for “*proberen*” (to test) and **V()** stands for “*verhogen*” (to increment) in Dutch

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.6

Semaphores Like Integers Except

- Semaphores are like integers, except
 - No negative values
 - Only operations allowed are P and V – can't read or write value, except to set it initially
 - Operations must be atomic
 - » Two P's together can't decrement value below zero
 - » Similarly, thread going to sleep in P won't miss wakeup from V – even if they both happen at same time
- Semaphore from railway analogy
 - Here is a semaphore initialized to 2 for resource control:



2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.7

Two Uses of Semaphores

- Mutual Exclusion (initial value = 1)
 - Also called "Binary Semaphore".
 - Can be used for mutual exclusion:
- Scheduling Constraints (initial value = 0)
 - Locks are fine for mutual exclusion, but what if you want a thread to wait for something?
 - Example: suppose you had to implement ThreadJoin which must wait for thread to terminate:

```
semaphore.P();
// Critical section goes here
semaphore.V();
```

```
Initial value of semaphore = 0
ThreadJoin {
    semaphore.P();
}
ThreadFinish {
    semaphore.V();
}
```

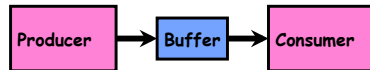
2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.8

Producer-consumer with a bounded buffer

- Problem Definition
 - Producer puts things into a shared buffer
 - Consumer takes them out
 - Need synchronization to coordinate producer/consumer
- Don't want producer and consumer to have to work in lockstep, so put a fixed-size buffer between them
 - Need to synchronize access to this buffer
 - Producer needs to wait if buffer is full
 - Consumer needs to wait if buffer is empty
- Example: Coke machine
 - Producer can put limited number of cokes in machine
 - Consumer can't take cokes out if machine is empty



2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.9

Correctness constraints for solution

- Correctness Constraints:
 - Consumer must wait for producer to fill slots, if empty (scheduling constraint)
 - Producer must wait for consumer to make room in buffer, if all full (scheduling constraint)
 - Only one thread can manipulate buffer queue at a time (mutual exclusion)
- General rule of thumb:
 - Use a separate semaphore for each constraint**
 - Semaphore fullSlots; // consumer's constraint
 - Semaphore emptySlots; // producer's constraint
 - Semaphore mutex; // mutual exclusion

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.10

Full Solution to Bounded Buffer

```
Semaphore fullSlots = 0; // Initially, no coke
Semaphore emptySlots = bufSize;
// Initially, num empty slots
Semaphore mutex = 1; // No one using machine

Producer(item) {
    emptySlots.P(); // Wait until space
    mutex.P(); // Wait until slot free
    Enqueue(item);
    mutex.V();
    fullSlots.V(); // Tell consumers there is
    // more coke
}

Consumer() {
    fullSlots.P(); // Check if there's a coke
    mutex.P(); // Wait until machine free
    item = Dequeue();
    mutex.V();
    emptySlots.V(); // tell producer need more
    return item;
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.11

Discussion about Solution

- Why asymmetry?
 - Producer does: `emptySlots.P()`, `fullSlots.V()`
 - Consumer does: `fullSlots.P()`, `emptySlots.V()`
- Is order of P's important?
 - Decrease # of empty slots
 - Increase # of occupied slots
- Is order of V's important?
 - Decrease # of occupied slots
 - Increase # of empty slots
- What if we have 2 producers or 2 consumers?
 - Do we need to change anything?

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.12

Motivation for Monitors and Condition Variables

- Semaphores are a huge step up; just think of trying to do the bounded buffer with only loads and stores
- Problem is that semaphores are dual purpose:
 - They are used for both mutex and scheduling constraints
 - Example: the fact that flipping of P's in bounded buffer gives deadlock is not immediately obvious. How do you prove correctness to someone?

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.13

Motivation for Monitors and Condition Variables

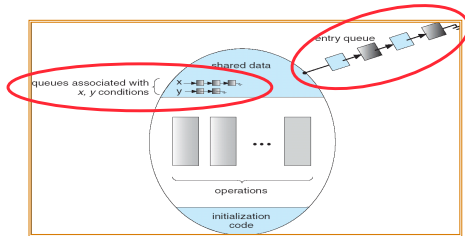
- Cleaner idea: Use *locks* for mutual exclusion and *condition variables* for scheduling constraints
- **Monitor**: a lock and zero or more condition variables for managing concurrent access to shared data
 - Some languages like Java provide this natively
 - Most others use actual locks and condition variables

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.14

Monitor with Condition Variables



- **Lock**: the lock provides mutual exclusion to shared data
 - Always acquire before accessing shared data structure
 - Always release after finishing with shared data
 - Lock initially free
- **Condition Variable**: a queue of threads waiting for something *inside* a critical section
 - Key idea: make it possible to go to sleep inside critical section by atomically releasing lock at time we go to sleep

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.15

Simple Monitor Example

- Here is an (infinite) synchronized queue

```

Lock lock;
Queue queue;

AddToQueue(item) {
    lock.Acquire(); // Lock shared data
    queue.enqueue(item); // Add item
    lock.Release(); // Release Lock
}

RemoveFromQueue() {
    lock.Acquire(); // Lock shared data
    item = queue.dequeue(); // Get next item or null
    lock.Release(); // Release Lock
    return(item); // Might return null
}
    
```

- Not very interesting use of “Monitor”
 - It only uses a lock with no condition variables
 - Cannot put consumer to sleep if no work!

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.16

Condition Variables

- **Condition Variable**: a queue of threads waiting for something *inside* a critical section
 - Key idea: allow sleeping inside critical section by atomically releasing lock at time we go to sleep
 - Contrast to semaphores: Can't wait inside critical section
- Operations:
 - `wait(&lock)`: Atomically release lock and go to sleep. Re-acquire lock later, before returning.
 - `signal()`: Wake up one waiter, if any
 - `broadcast()`: Wake up all waiters
- Rule: Must hold lock when doing condition variable ops!

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.17

Complete Monitor Example (with condition variable)

- Here is an (infinite) synchronized queue

```

Lock lock;
Condition dataready;
Queue queue;

AddToQueue(item) {
    lock.Acquire(); // Get Lock
    queue.enqueue(item); // Add item
    dataready.signal(); // Signal any waiters
    lock.Release(); // Release Lock
}

RemoveFromQueue() {
    lock.Acquire(); // Get Lock
    while (queue.isEmpty()) {
        dataready.wait(&lock); // If nothing, sleep
    }
    item = queue.dequeue(); // Get next item
    lock.Release(); // Release Lock
    return(item);
}
    
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.18

Mesa vs. Hoare monitors

- Need to be careful about precise definition of signal and wait. Consider a piece of our dequeue code:

```

while (queue.isEmpty()) {
    dataready.wait(&lock); // If nothing, sleep
}
item = queue.dequeue(); // Get next item
- Why didn't we do this?
if (queue.isEmpty()) {
    dataready.wait(&lock); // If nothing, sleep
}
item = queue.dequeue(); // Get next item
    
```

- Answer: depends on the type of scheduling
 - Hoare-style
 - Mesa-style

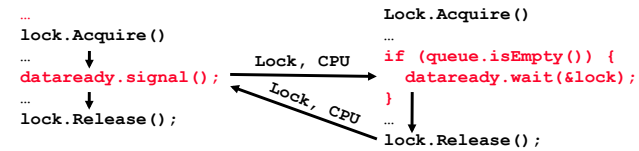
2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.19

Hoare monitors

- Signaler gives up lock, CPU to waiter; waiter runs immediately
- Waiter gives up lock, processor back to signaler when it exits critical section or if it waits again
- Most textbooks



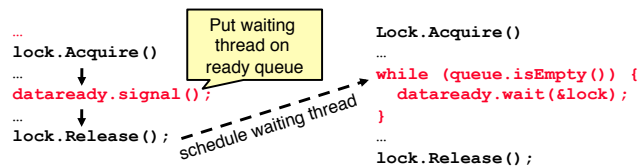
2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.20

Mesa monitors

- Signaler keeps lock and processor
- Waiter placed on ready queue with no special priority
- Practically, need to check condition again after wait
- Most real operating systems



2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.21

5min Break

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.22

Readers/Writers Problem

- Motivation: Consider a shared database
 - Two classes of users:
 - » Readers – never modify database
 - » Writers – read and modify database
 - Is using a single lock on the whole database sufficient?
 - » Like to have many readers at the same time
 - » Only one writer at a time

2/1/12 Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 5.23

Basic Readers/Writers Solution

- Correctness Constraints:
 - Readers can access database when no writers
 - Writers can access database when no readers or writers
 - Only one thread manipulates state variables at a time
- Basic structure of a solution:
 - Reader()
 - Wait until no writers
 - Access data base
 - Check out – wake up a waiting writer
 - Writer()
 - Wait until no active readers or writers
 - Access database
 - Check out – wake up waiting readers or writer
 - State variables (Protected by a lock called “lock”):
 - » int AR: Number of active readers; initially = 0
 - » int WR: Number of waiting readers; initially = 0
 - » int AW: Number of active writers; initially = 0
 - » int WW: Number of waiting writers; initially = 0
 - » Condition okToRead = NIL
 - » Condition okToWrite = NIL

2/1/12 Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 5.24

Code for a Reader

```

Reader() {
  // First check self into system
  lock.Acquire();
  while ((AW + WW) > 0) { // Is it safe to read?
    WR++; // No. Writers exist
    okToRead.wait(&lock); // Sleep on cond var
    WR--; // No longer waiting
  }
  AR++; // Now we are active!
  lock.release();
  // Perform actual read-only access
  AccessDatabase(ReadOnly);
  // Now, check out of system
  lock.Acquire();
  AR--; // No longer active
  if (AR == 0 && WW > 0) // No other active readers
    okToWrite.signal(); // Wake up one writer
  lock.Release();
}

```

2/1/12 Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 5.25

Code for a Writer

```

Writer() {
  // First check self into system
  lock.Acquire();
  while ((AW + AR) > 0) { // Is it safe to write?
    WW++; // No. Active users exist
    okToWrite.wait(&lock); // Sleep on cond var
    WW--; // No longer waiting
  }
  AW++; // Now we are active!
  lock.release();
  // Perform actual read/write access
  AccessDatabase(ReadWrite);
  // Now, check out of system
  lock.Acquire();
  AW--; // No longer active
  if (WW > 0) { // Give priority to writers
    okToWrite.signal(); // Wake up one writer
  } else if (WR > 0) { // Otherwise, wake reader
    okToRead.broadcast(); // Wake all readers
  }
  lock.Release();
}

```

2/1/12 Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 5.26

Simulation of Readers/Writers Solution

- Use an example to simulate the solution
- Consider the following sequence of operators:
 - R1, R2, W1, R3
- Initially: AR = 0, WR = 0, AW = 0, WW = 0

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.27

Simulation of Readers/Writers Solution

- R1 comes along
- AR = 0, WR = 0, AW = 0, WW = 0

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.28

Simulation of Readers/Writers Solution

- R1 comes along
- AR = 0, WR = 0, AW = 0, WW = 0

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.29

Simulation of Readers/Writers Solution

- R1 comes along
- AR = 1, WR = 0, AW = 0, WW = 0

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.30

Simulation of Readers/Writers Solution

- R1 comes along
- AR = 1, WR = 0, AW = 0, WW = 0

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.31

Simulation of Readers/Writers Solution

- R1 comes along
- AR = 1, WR = 0, AW = 0, WW = 0

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.32

Simulation of Readers/Writers Solution

- R2 comes along
- AR = 1, WR = 0, AW = 0, WW = 0

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.33

Simulation of Readers/Writers Solution

- R2 comes along
- AR = 1, WR = 0, AW = 0, WW = 0

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.34

Simulation of Readers/Writers Solution

- R2 comes along
- AR = 2, WR = 0, AW = 0, WW = 0

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.35

Simulation of Readers/Writers Solution

- R2 comes along
- AR = 2, WR = 0, AW = 0, WW = 0

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.36

Simulation of Readers/Writers Solution

- R2 comes along
- AR = 2, WR = 0, AW = 0, WW = 0

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
}
1 Assume readers take a while to access database
    Situation: Locks released, only AR is non-zero
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.37

Simulation of Readers/Writers Solution

- W1 comes along (R1 and R2 are still accessing dbase)
- AR = 2, WR = 0, AW = 0, WW = 0

```
Writer() {
    lock.Acquire();
    while ((AW + AR) > 0) { // Is it safe to write?
        WW++; // No. Active users exist
        okToWrite.wait(&lock); // Sleep on cond var
        WW--; // No longer waiting
    }
    AW++;
    lock.release();

    AccessDbase(ReadWrite);

    lock.Acquire();
    AW--;
    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.38

Simulation of Readers/Writers Solution

- W1 comes along (R1 and R2 are still accessing dbase)
- AR = 2, WR = 0, AW = 0, WW = 0

```
Writer() {
    lock.Acquire();
    while ((AW + AR) > 0) { // Is it safe to write?
        WW++; // No. Active users exist
        okToWrite.wait(&lock); // Sleep on cond var
        WW--; // No longer waiting
    }
    AW++;
    lock.release();

    AccessDbase(ReadWrite);

    lock.Acquire();
    AW--;
    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.39

Simulation of Readers/Writers Solution

- W1 comes along (R1 and R2 are still accessing dbase)
- AR = 2, WR = 0, AW = 0, WW = 1

```
Writer() {
    lock.Acquire();
    while ((AW + AR) > 0) { // Is it safe to write?
        WW++; // No. Active users exist
        okToWrite.wait(&lock); // Sleep on cond var
        WW--; // No longer waiting
    }
    AW++;
    lock.release();

    AccessDbase(ReadWrite);

    lock.Acquire();
    AW--;
    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.40

Simulation of Readers/Writers Solution

- W1 comes along (R1 and R2 are still accessing dbase)
- AR = 2, WR = 0, AW = 0, WW = 1

```
Writer() {
    lock.Acquire();
    while ((AW + AR) > 0) { // Is it safe to write?
        WW++; // No. Active users exist
        okToWrite.wait(&lock); // Sleep on cond var
        WW--; // No longer waiting
    }
    AW++;
    lock.release();

    AccessDbase(ReadWrite);

    lock.Acquire();
    AW--;
    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.Release();
}
W1 cannot start because of readers, so goes to sleep
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.41

Simulation of Readers/Writers Solution

- R3 comes along (R1, R2 accessing dbase, W1 waiting)
- AR = 2, WR = 0, AW = 0, WW = 1

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.42

Simulation of Readers/Writers Solution

- R3 comes along (R1, R2 accessing dbase, W1 waiting)
- AR = 2, WR = 0, AW = 0, WW = 1

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.43

Simulation of Readers/Writers Solution

- R3 comes along (R1, R2 accessing dbase, W1 waiting)
- AR = 2, WR = 1, AW = 0, WW = 1

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.44

Simulation of Readers/Writers Solution

- R3 comes along (R1, R2 accessing dbase, W1 waiting)
- AR = 2, WR = 1, AW = 0, WW = 1

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
```

Status:

- R1 and R2 still reading
- W1 and R3 waiting on okToWrite and okToRead, respectively

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.45

Simulation of Readers/Writers Solution

- R2 finishes (R1 accessing dbase, W1, R3 waiting)
- AR = 2, WR = 1, AW = 0, WW = 1

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.46

Simulation of Readers/Writers Solution

- R2 finishes (R1 accessing dbase, W1, R3 waiting)
- AR = 1, WR = 1, AW = 0, WW = 1

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.47

Simulation of Readers/Writers Solution

- R2 finishes (R1 accessing dbase, W1, R3 waiting)
- AR = 1, WR = 1, AW = 0, WW = 1

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.48

Simulation of Readers/Writers Solution

- R2 finishes (R1 accessing dbase, W1, R3 waiting)
- AR = 1, WR = 1, AW = 0, WW = 1

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.49

Simulation of Readers/Writers Solution

- R1 finishes (W1, R3 waiting)
- AR = 1, WR = 1, AW = 0, WW = 1

```
Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.50

Simulation of Readers/Writers Solution

- R1 finishes (W1, R3 waiting)
- AR = 0, WR = 1, AW = 0, WW = 1

```

Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
    
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.51

Simulation of Readers/Writers Solution

- R1 finishes (W1, R3 waiting)
- AR = 0, WR = 1, AW = 0, WW = 1

```

Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
    
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.52

Simulation of Readers/Writers Solution

- R1 finishes (W1, R3 waiting)
- AR = 0, WR = 1, AW = 0, WW = 1

```

Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
    
```

All reader finished, signal writer – note R3 still waiting

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.53

Simulation of Readers/Writers Solution

- W1 gets signal (R3 still waiting)
- AR = 0, WR = 1, AW = 0, WW = 1

```

Writer() {
    lock.Acquire();
    while ((AW + AR) > 0) { // Is it safe to write?
        WW++; // No. Active users exist
        okToWrite.wait(&lock); // Sleep on cond var
        WW--; // No longer waiting
    }
    AR++;
    lock.release();

    AccessDbase(ReadWrite);

    lock.Acquire();
    AW--;
    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.Release();
}
    
```

Got signal from R1

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.54

Simulation of Readers/Writers Solution

- W1 gets signal (R3 still waiting)
- AR = 0, WR = 1, AW = 0, WW = 0

```
Writer() {
    lock.Acquire();
    while ((AW + AR) > 0) { // Is it safe to write?
        WW++; // No. Active users exist
        okToWrite.wait(&lock); // Sleep on cond var
        WW--; // No longer waiting
    }
    AW++;
    lock.release();

    AccessDbase(ReadWrite);

    lock.Acquire();
    AW--;
    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.55

Simulation of Readers/Writers Solution

- W1 gets signal (R3 still waiting)
- AR = 0, WR = 1, AW = 1, WW = 0

```
Writer() {
    lock.Acquire();
    while ((AW + AR) > 0) { // Is it safe to write?
        WW++; // No. Active users exist
        okToWrite.wait(&lock); // Sleep on cond var
        WW--; // No longer waiting
    }
    AW++;
    lock.release();

    AccessDbase(ReadWrite);

    lock.Acquire();
    AW--;
    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.56

Simulation of Readers/Writers Solution

- W1 gets signal (R3 still waiting)
- AR = 0, WR = 1, AW = 1, WW = 0

```
Writer() {
    lock.Acquire();
    while ((AW + AR) > 0) { // Is it safe to write?
        WW++; // No. Active users exist
        okToWrite.wait(&lock); // Sleep on cond var
        WW--; // No longer waiting
    }
    AW++;
    lock.release();

    AccessDbase(ReadWrite);

    lock.Acquire();
    AW--;
    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.57

Simulation of Readers/Writers Solution

- W1 gets signal (R3 still waiting)
- AR = 0, WR = 1, AW = 0, WW = 0

```
Writer() {
    lock.Acquire();
    while ((AW + AR) > 0) { // Is it safe to write?
        WW++; // No. Active users exist
        okToWrite.wait(&lock); // Sleep on cond var
        WW--; // No longer waiting
    }
    AW++;
    lock.release();

    AccessDbase(ReadWrite);

    lock.Acquire();
    AW--;
    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.Release();
}
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.58

Simulation of Readers/Writers Solution

- W1 gets signal (R3 still waiting)
- AR = 0, WR = 1, AW = 0, WW = 0

```

Writer() {
    lock.Acquire();
    while ((AW + AR) > 0) { // Is it safe to write?
        WW++; // No. Active users exist
        okToWrite.wait(&lock); // Sleep on cond var
        WW--; // No longer waiting
    }
    AW++;
    lock.release();

    AccessDbase(ReadWrite);

    lock.Acquire();
    AW--;
    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.Release();
}

```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.59

Simulation of Readers/Writers Solution

- W1 gets signal (R3 still waiting)
- AR = 0, WR = 1, AW = 0, WW = 0

```

Writer() {
    lock.Acquire();
    while ((AW + AR) > 0) { // Is it safe to write?
        WW++; // No. Active users exist
        okToWrite.wait(&lock); // Sleep on cond var
        WW--; // No longer waiting
    }
    AW++;
    lock.release();

    AccessDbase(ReadWrite);

    lock.Acquire();
    AW--;
    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.Release();
}

```

No waiting writer, signal reader R3

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.60

Simulation of Readers/Writers Solution

- R1 finishes (W1, R3 waiting)
- AR = 0, WR = 1, AW = 0, WW = 0

```

Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    ; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}

```

Got signal from W1

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.61

Simulation of Readers/Writers Solution

- R1 finishes (W1, R3 waiting)
- AR = 0, WR = 0, AW = 0, WW = 0

```

Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}

```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.62

Simulation of Readers/Writers Solution

- R1 finishes (W1, R3 waiting)
- AR = 0, WR = 0, AW = 0, WW = 0

```

Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
    
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.63

Simulation of Readers/Writers Solution

- R1 finishes (W1, R3 waiting)
- AR = 0, WR = 0, AW = 0, WW = 0

```

Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
    
```

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.64

Simulation of Readers/Writers Solution

- R1 finishes (W1, R3 waiting)
- AR = 0, WR = 0, AW = 0, WW = 0

```

Reader() {
    lock.Acquire();
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        okToRead.wait(&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDbase(ReadOnly);

    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}
    
```

DONE!

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.65

Read/Writer Questions

```

Reader() {
    // check into system
    lock.Acquire();
    while ((AW + WW) > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // read-only
    AccessDbase(ReadOnly);

    // check out of system
    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.signal();
    lock.Release();
}

Writer() {
    // check into system
    lock.Acquire();
    while ((AW + AR) > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // read/write access
    AccessDbase(ReadWrite);

    // check out of system
    lock.Acquire();
    AW--;
    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.Release();
}
    
```

What if we remove this line?

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.66

Read/Writer Questions

```

Reader() {
    // check into system
    lock.Acquire();
    while ((AW + WW) > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // read-only
    AccessDbase(ReadOnly);

    // check out of system
    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okToWrite.broadcast();
    lock.Release();
}

Writer() {
    // check into system
    lock.Acquire();
    while ((AW + AR) > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // read/write access
    AccessDbase(ReadWrite);

    // check out of system
    lock.Acquire();
    AW--;
    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.Release();
}

```

What if we turn signal to broadcast?

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.67

Read/Writer Questions

```

Reader() {
    // check into system
    lock.Acquire();
    while ((AW + WW) > 0) {
        WR++;
        okContinue.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // read-only access
    AccessDbase(ReadOnly);

    // check out of system
    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okContinue.signal();
    lock.Release();
}

Writer() {
    // check into system
    lock.Acquire();
    while ((AW + AR) > 0) {
        WW++;
        okContinue.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // read/write access
    AccessDbase(ReadWrite);

    // check out of system
    lock.Acquire();
    AW--;
    if (WW > 0) {
        okContinue.signal();
    } else if (WR > 0) {
        okContinue.broadcast();
    }
    lock.Release();
}

```

What if we turn okToWrite and okToRead into okContinue?

2/1/12

Read/Writer Questions

```

Reader() {
    // check into system
    lock.Acquire();
    while ((AW + WW) > 0) {
        WR++;
        okContinue.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // read-only access
    AccessDbase(ReadOnly);

    // check out of system
    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okContinue.signal();
    lock.Release();
}

Writer() {
    // check into system
    lock.Acquire();
    while ((AW + AR) > 0) {
        WW++;
        okContinue.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // read/write access
    AccessDbase(ReadWrite);

    // check out of system
    lock.Acquire();
    AW--;
    if (WW > 0) {
        okContinue.signal();
    } else if (WR > 0) {
        okContinue.broadcast();
    }
    lock.Release();
}

```

- R1 arrives
- W1, R2 arrive while R1 still reading → W1 and R2 both wait for R1 to finish
- R1 signals R2

2/1/12

Read/Writer Questions

```

Reader() {
    // check into system
    lock.Acquire();
    while ((AW + WW) > 0) {
        WR++;
        okContinue.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // read-only access
    AccessDbase(ReadOnly);

    // check out of system
    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
        okContinue.broadcast();
    lock.Release();
}

Writer() {
    // check into system
    lock.Acquire();
    while ((AW + AR) > 0) {
        WW++;
        okContinue.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // read/write access
    AccessDbase(ReadWrite);

    // check out of system
    lock.Acquire();
    AW--;
    if (WW > 0) {
        okContinue.signal();
    } else if (WR > 0) {
        okContinue.broadcast();
    }
    lock.Release();
}

```

Need to change to broadcast!

2/1/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 5.70

Summary

- Semaphores
 - Generalized locks
 - Two operations: `P()`, `V()`
- Monitors: A lock plus one or more condition variables
 - Always acquire lock before accessing shared data
 - Use condition variables to wait inside critical section
 - » Three Operations: `Wait()`, `Signal()`, and `Broadcast()`
- Readers/Writers
 - Readers can access database when no writers
 - Writers can access database when no readers
 - Only one thread manipulates state variables at a time