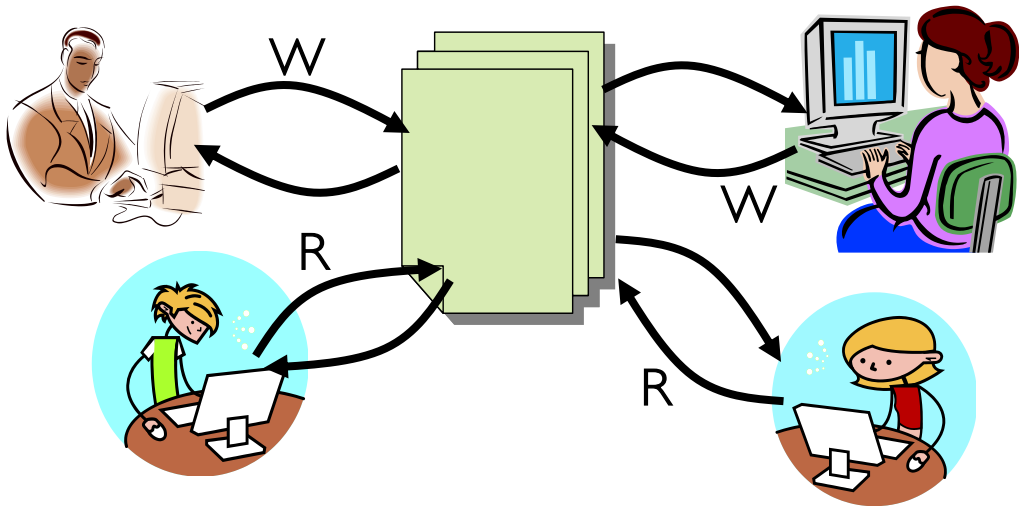
ECE 350 Real-time Operating Systems

Tutorial
Reader/Writer Lock

Prof. Seyed Majid Zahedi

https://ece.uwaterloo.ca/~smzahedi

Readers/Writers Lock



- Motivation: consider shared database with two classes of users
  - Readers: never modify database
  - Writers: read and modify database
- Database can have many readers at the same time
- But there can be only one writer active at a time

Properties of Readers/Writers Lock

- Common variant of mutual exclusion
  - One writer at a time, if no readers
  - Many readers, if no writer

Thread 1 \ Thread 2	Writer	Reader
Writer	NO!	NO!
Reader	NO!	OK!

- Correctness constraints
  - Readers can read when no writers
  - Writers can read/write when no readers or writers
  - Only one thread manipulates *state of the lock* at a time

Readers/Writers Lock Class

```
class ReadWriteLock {
    private:
        Lock lock;                  // needed to change state vars
        CV okToRead                 // CV for readers
        CV okToWrite;               // CV for writers
        int AW = 0;                 // # of active writers
        int AR = 0;                 // # of active readers
        int WW = 0;                 // # of waiting writers
        int WR = 0;                 // # of waiting readers

    public:
        void acquireRL();
        void releaseRL();
        void acquireWL();
        void releaseWL();
}
```

Readers/Writers Lock Design Pattern

```
read() {
    lock.acquireRL();

    // Read shared state

    lock.releaseRL();
}
```

```
write() {
    lock.acquireWL();

    // Read/write shared state

    lock.releaseWL();
}
```

Readers/Writers Lock Implementation

```
acquireRL() {
    lock.acquire();                 // Need lock to change state vars
    while (AW + WW > 0) {           // Is it safe to read?
        WR++;                       // No! add to # of waiting readers
        okToRead.wait(&lock);       // Wait on condition variable
        WR--;                       // No longer waiting
    }
    AR++;                            // Now we are active again
    lock.release();
}

releaseRL() {
    lock.acquire();
    AR--;                           // No longer active
    if (AR == 0 && WW > 0)          // If no active reader,
        okToWrite.signal();         //  wake up waiting writer
    lock.release();
}
```

Why release lock here?

Lock is acquired to change internal state of R/W lock

Readers/Writers Lock Implementation (cont.)

```
acquireWL() {
    lock.acquire();
    while (AW + AR > 0) {          // is it safe to write?
        WW++;                      // No! add to # of waiting writers
        okToWrite.wait(&lock);     // Wait on condition variable
        WW--;                      // No longer waiting
    }
    AW++;                          // Now we are active again
    lock.release();
}


releaseWL() {
    lock.acquire();
    AW--;                          // No longer active
    if (WW > 0) {                  // Give priority to writers
        okToWrite.signal();        // Wake up a waiting writer
    } else if (WR > 0) {           // If there are waiting readers,
        okToRead.broadcast();      //  wake them all up
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- Consider following sequence of arrivals: R1, R2, W1, R3

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- R1 comes – `AR = 0, WR = 0, AW = 0, WW = 0`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

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

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R1 comes – `AR = 0, WR = 0, AW = 0, WW = 0`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

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

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R1 comes – `AR = 1, WR = 0, AW = 0, WW = 0`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R1 comes – `AR = 1, WR = 0, AW = 0, WW = 0`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R2 comes – `AR = 1, WR = 0, AW = 0, WW = 0`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R2 comes – `AR = 1, WR = 0, AW = 0, WW = 0`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R2 comes – `AR = 1, WR = 0, AW = 0, WW = 0`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

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

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R2 comes – `AR = 2, WR = 0, AW = 0, WW = 0`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R2 comes – `AR = 2, WR = 0, AW = 0, WW = 0`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Assume readers take a while to access database
Situation: Locks released, only AR is non-zero

Simulation of Readers/Writers Lock

- W1 comes – AR = 2, WR = 0, AW = 0, WW = 0

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- W1 comes – AR = 2, WR = 0, AW = 0, WW = 0

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- W1 comes – AR = 2, WR = 0, AW = 0, WW = 0

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- W1 comes – AR = 2, WR = 0, AW = 0, WW = 1

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- W1 comes – `AR = 2, WR = 0, AW = 0, WW = 1`

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

W1 cannot start because of readers, so it releases lock and goes to sleep

Simulation of Readers/Writers Lock

- R3 comes – `AR = 2, WR = 0, AW = 0, WW = 1`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R3 comes – `AR = 2, WR = 0, AW = 0, WW = 1`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R3 comes – `AR = 2, WR = 0, AW = 0, WW = 1`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R3 comes – AR = 2, WR = 1, AW = 0, WW = 1

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R3 comes – `AR = 2, WR = 1, AW = 0, WW = 1`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Status:

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

Simulation of Readers/Writers Lock

- R2 is done reading – `AR = 2, WR = 1, AW = 0, WW = 1`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R2 is done reading – `AR = 1, WR = 1, AW = 0, WW = 1`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R2 is done reading – `AR = 1, WR = 1, AW = 0, WW = 1`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R2 is done reading – `AR = 1, WR = 1, AW = 0, WW = 1`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R1 is done reading – `AR = 1, WR = 1, AW = 0, WW = 1`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R1 is done reading – `AR = 0, WR = 1, AW = 0, WW = 1`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R1 is done reading – `AR = 0, WR = 1, AW = 0, WW = 1`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R1 is done reading – `AR = 0, WR = 1, AW = 0, WW = 1`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

All readers are finished, R1 signals waiting writer – note, R3 is still waiting

Simulation of Readers/Writers Lock

- R1 is done reading – `AR = 0, WR = 1, AW = 0, WW = 1`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- W1 gets a signal – `AR = 0, WR = 1, AW = 0, WW = 1`

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

W1 gets signal from R1

Simulation of Readers/Writers Lock

- W1 gets a signal – AR = 0, WR = 1, AW = 0, WW = 0

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- W1 gets a signal – AR = 0, WR = 1, AW = 1, WW = 0

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- W1 gets a signal – `AR = 0, WR = 1, AW = 1, WW = 0`

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- W1 gets a signal – `AR = 0, WR = 1, AW = 1, WW = 0`

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- W1 is done – `AR = 0, WR = 1, AW = 1, WW = 0`

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- W1 is done – AR = 0, WR = 1, AW = 0, WW = 0

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- W1 is done – `AR = 0, WR = 1, AW = 0, WW = 0`

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- W1 is done – `AR = 0, WR = 1, AW = 0, WW = 0`

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- W1 is done – `AR = 0, WR = 1, AW = 0, WW = 0`

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

No waiting writer, so only signal R3

Simulation of Readers/Writers Lock

- W1 is done – `AR = 0, WR = 1, AW = 0, WW = 0`

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Simulation of Readers/Writers Lock

- R3 gets a signal – `AR = 0, WR = 1, AW = 0, WW = 0`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

R3 gets signal from W3

Simulation of Readers/Writers Lock

- R3 gets a signal – `AR = 0, WR = 0, AW = 0, WW = 0`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R3 gets a signal – `AR = 1, WR = 0, AW = 0, WW = 0`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Simulation of Readers/Writers Lock

- R3 finishes – `AR = 0, WR = 0, AW = 0, WW = 0`

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

Readers/Writers Lock Questions

```
read() {
    lock.acquire();
    while (AW + WW > 0) {
        WR++;
        okToRead.wait(&lock);
        WR--;
    }
    AR++;
    lock.release();

    // Read

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

What if we remove this line?

```
write() {
    lock.acquire();
    while (AW + AR > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();

    // Read and Write

    lock.acquire();
    AW--;

    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

It works but it's inefficient, writer wakes up and goes to sleep again when it's not save to write

Readers/Writers Lock Questions

```
read() {
   lock.acquire();
   while (AW + WW > 0) {
      WR++;
      okToRead.wait(&lock);
      WR--;
   }
   AR++;
   lock.release();

   // Read

   lock.acquire();
   AR--;
   if (AR == 0 && WW > 0)
      okToWrite.broadcast();
   lock.release();
}
```

What if we turn `signal()` to `broadcast()`?

```
write() {
   lock.acquire();
   while (AW + AR > 0) {
      WW++;
      okToWrite.wait(&lock);
      WW--;
   }
   AW++;
   lock.release();

   // Read and Write

   lock.acquire();
   AW--;

   if (WW > 0) {
      okToWrite.signal();
   } else if (WR > 0) {
      okToRead.broadcast();
   }
   lock.release();
}
```

It works but it's inefficient to wake up all writers only for one to becomes active

Readers/Writers Lock Questions

```
read() {
   lock.acquire();
   while (AW + WW > 0) {
      WR++;
      okToContinue.wait(&lock);
      WR--;
   }
   AR++;
   lock.release();

   // Read

   lock.acquire();
   AR--;
   if (AR == 0 && WW > 0)
      okToContinue.signal();
   lock.release();
}
```

What if we turn **okToWrite** and **okToRead** into **okContinue**?

Signal could be delivered to wrong thread (reader) and get waisted!

```
write() {
   lock.acquire();
   while (AW + AR > 0) {
      WW++;
      okToContinue.wait(&lock);
      WW--;
   }
   AW++;
   lock.release();

   // Read and Write

   lock.acquire();
   AW--;

   if (WW > 0) {
      okToContinue.signal();
   } else if (WR > 0) {
      okToContinue.broadcast();
   }
   lock.release();
}
```

Readers/Writers Lock Questions

```
read() {
   lock.acquire();
   while (AW + WW > 0) {
      WR++;
      okToContinue.wait(&lock);
      WR--;
   }
   AR++;
   lock.release();

   // Read

   lock.acquire();
   AR--;
   if (AR == 0 && WW > 0)
      okToContinue.broadcast();
   lock.release();
}
```

```
write() {
   lock.acquire();
   while (AW + AR > 0) {
      WW++;
      okToContinue.wait(&lock);
      WW--;
   }
   AW++;
   lock.release();

   // Read and Write

   lock.acquire();
   AW--;

   if (WW > 0) {
      okToContinue.broadcast();
   } else if (WR > 0) {
      okToContinue.broadcast();
   }
   lock.release();
}
```

Does changing **signal()** to **broadcast()** solve the problem?

Yes, but it's inefficient to wake up all threads for only one to becomes active

Readers/Writers Lock Questions

```
read() {
   lock.acquire();
   while (AW + WW > 0) {
      WR++;
      okToRead.wait(&lock);
      WR--;
   }
   AR++;
   lock.release();

   // Read

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

Can readers starve?

Yes: writers take priority

```
write() {
   lock.acquire();
   while (AW + AR > 0) {
      WW++;
      okToWrite.wait(&lock);
      WW--;
   }
   AW++;
   lock.release();

   // Read and Write

   lock.acquire();
   AW--;

   if (WW > 0) {
      okToWrite.signal();
   } else if (WR > 0) {
      okToRead.broadcast();
   }
   lock.release();
}
```

Readers/Writers Lock Questions

```
read() {
   lock.acquire();
   while (AW + WW > 0) {
      WR++;
      okToRead.wait(&lock);
      WR--;
   }
   AR++;
   lock.release();

   // Read

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

```
write() {
   lock.acquire();
   while (AW + AR > 0) {
      WW++;
      okToWrite.wait(&lock);
      WW--;
   }
   AW++;
   lock.release();

   // Read and Write

   lock.acquire();
   AW--;

   if (WW > 0) {
      okToWrite.signal();
   } else if (WR > 0) {
      okToRead.broadcast();
   }
   lock.release();
}
```

Can writers starve?

Yes: a waiting writer may not be able to proceed if another writer slips in between signal and wakeup

Readers/Writers Lock Without Writer Starvation (Take 1)

check also for waiting writers

```
acquireWL() {
    lock.acquire();
    while (AW + AR + WW > 0) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();
}
```

- Does this work?
  - No! If there `WW` is more than zero, then no waiting writer can successfully proceed

Readers/Writers Lock Without Writer Starvation (Take 2)

Idea: keep track of writers' waiting order, allow writer with longest waiting time to proceed

Does this work?

```
numWriters = 0;
nextToGo = 1;

acquireWL() {
    lock.acquire();
    myPos = numWriters++;
    while (AW + AR > 0 ||
              myPos > nextToGo) {
        WW++;
        okToWrite.wait(&lock);
        WW--;
    }
    AW++;
    lock.release();
}
```

No, Signal can wake up a wrong writer and get waisted!

```
releaseWL() {
    lock.acquire();
    AW--;
    nextToGo++;
    if (WW > 0) {
        okToWrite.signal();
    } else if (WR > 0) {
        okToRead.broadcast();
    }
    lock.release();
}
```

Inefficient solution is to change `signal()` to `broadcast()`

Readers/Writers Lock Without Writer Starvation (Take 3)

Idea for efficient solution: have separate CV for each waiting writer and put CV's in ordered queue

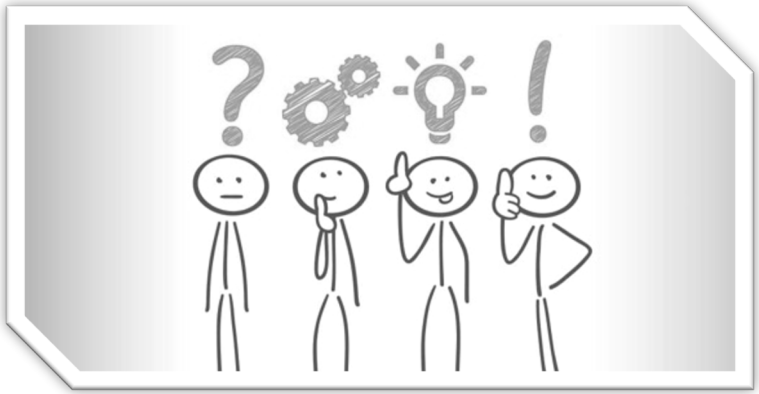
```
numWriters = 0;
nextToGo = 1;

acquireWL() {
   lock.acquire();
   myPos = numWriters++;
   myCV = new CV();
   queue.enqueue(myCV);
   while (AW + AR > 0 ||
             myPos > nextToGo) {
      WW++;
      myCV.wait(&lock);
      WW--;
   }
   AW++;
   queue.dequeue();
   lock.release();
}
```

```
releaseWL() {
   lock.acquire();
   AW--;
   nextToGo++;
   if (WW > 0) {
      queue.head().signal();
   } else if (WR > 0) {
      okToRead.broadcast();
   }
   lock.release();
}

releaseRL() {
   lock.acquire();
   AR--;
   if (AR == 0 && WW > 0)
      queue.head().signal();
   lock.release();
}
```

Questions?



Acknowledgment

- Slides by courtesy of Anderson, Culler, Stoica, Silberschatz, Joseph, and Canny