- Allow multiple readers to execute in the critical section at once.
- Clearance mutual exclusion for writers.
- Each read or write of the shared data must happen within a critical section.

Correctness criteria:

- How do we control access to the object to permit this protocol?
- Allow only one writer at any point
- Want many readers reading the object at once

Using a single lock on the data object is overly restrictive

Write: read data and modify it
Read: read data, never modify it

Classes:
An object is shared among many threads, each belonging to one of two

Today: Synchronization for Readers/Writers Problem

Semaphores are useful for mutual exclusion, progress and bounded waiting

Set of resources counting semaphore: useful for granting mutually exclusive access for a

Binary or Mutex Semaphore: grants mutual exclusive access to a

Semaphore S supports two atomic operations:
- S->signal(): release the semaphore, wake up a process if one is waiting for S.
- S->wait(): get a semaphore, wait if busy Semaphore S is available.
Readers/Writers Problem

{ mutex->wait();
  enable writers //!
  mutex->wait();
} if (readers == 0)
{
  reader done
  mutex->wait();
  enable mutual exclusion

  mutex->wait();
  perform read
} if (readers == 1)
{
  another reader
  mutex->wait();
  enable mutual exclusion

  (readers) += 1
}

ReadWrite::Read
{
  any writers or readers?
  enable others

  mutex->wait();
  perform write
}

ReadWrite::Write
{
  if (!readers)
  writer or reader

  writer->value = 1
  mutex->wait();
  readers = 1
}

ReadWrite::ReadWrite
{
  Semaphore write;
  Semaphore mutex;

  int readers;
  counts readers
  public:
    void Write()
    { volatile Write();
    }
    void Read()
    { volatile Reader();
    }
}

Readers/Writers Problem
Read ()
Write ()

Read (): MI
Write (): R1: MI

Read (): R2:

Readers/Writers: Scenario 2

Readers/Writers: Scenario 1
Let a writer enter its critical section as soon as possible. 

Alternative desirable semantics:

5. Does this solution guarantee all threads will make progress?

4. If a writer exists and a reader goes next, then all readers that are waiting will fall

3. When a writer exists, if there is both a reader and writer waiting, which goes next

2. The last reader to exit signals a waiting writer.

1. The first reader blocks if there is a writer, any other readers who try to enter block

Implementation notes:

Readers/ Writers Solution Discussion

Read()  Write()  Read()  Write()

R1:  R2:  R1:  R2:
Readers/Writers Solution: Favoring Writers

Readers/Writers Solution: Favoring Writers

Readers/Writers Solution: Favoring Writers
Readers/ Writers: Scenario 5

Read
Write
R2: M2: W2
R1:

Readers/ Writers: Scenario 4

Read
Write
R2: M2: W2
R1:
Problem

Reading assignment: read OSC ch 6, pages 175-177 for a solution to this

After eating, put down both chopsticks and go back to thinking

- Block if neighbor has already picked up a chopstick
- Eating: need two chopsticks, try to pick up two closest chopsticks
- Thinking: do nothing

Share a circular table with five chopsticks

Five philosophers, each either eats or thinks

```
Other Synchronization Problems: Dining Philosophers
```

```
Write

Read

Write

Write

Read

Read

M1: M2: R1:

Scenario 6
```
Summary

- Readers/writers problem:
  - Allow multiple readers to concurrently access a data
  - Allow only one writer at a time
- Two possible solutions using semaphores
- Starvation is possible in either case!