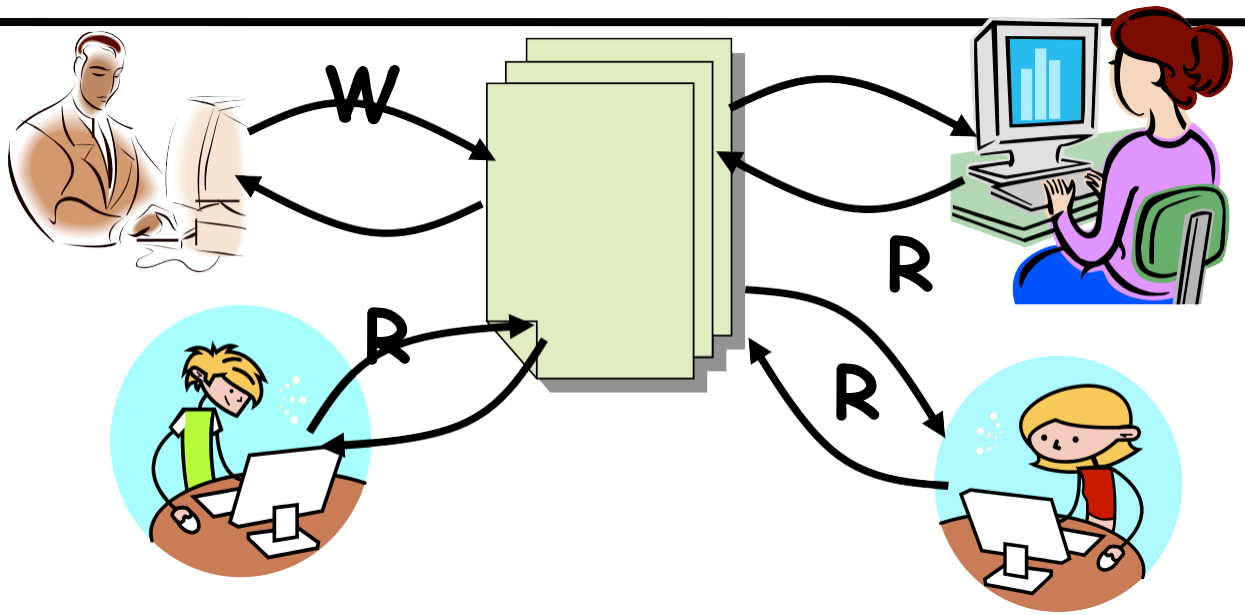
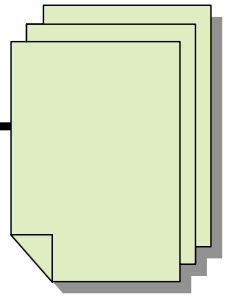
Reader/Writer Synchronization

Readers/Writers Problem

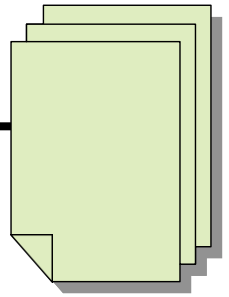


- Motivation: Consider a shared database
  - Two classes of users:
    - » Readers – never modify database
    - » Writers – read and modify database
  - Is using a single lock on the whole database sufficient?
    - » Like to have many readers at the same time
    - » Only one writer at a time

Basic Readers/Writers Solution

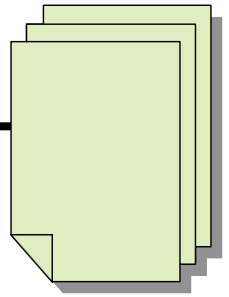


Basic Readers/Writers Solution



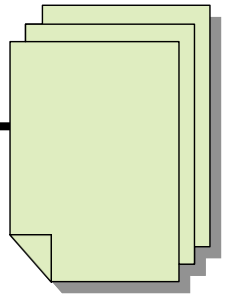
- Correctness Constraints:
  - Readers can access database when no writers
  - Writers can access database when no readers
  - Only one thread manipulates state variables at a time

Basic Readers/Writers Solution



- Correctness Constraints:
  - Readers can access database when no writers
  - Writers can access database when no readers
  - Only one thread manipulates state variables at a time
- Basic structure of a solution:

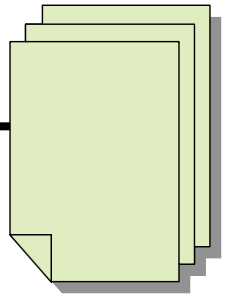
Basic Readers/Writers Solution



- Correctness Constraints:
  - Readers can access database when no writers
  - Writers can access database when no readers
  - 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
```

Basic Readers/Writers Solution



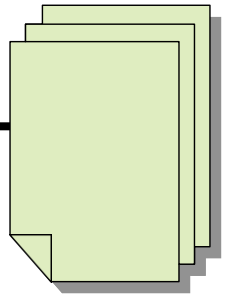
- Correctness Constraints:
  - Readers can access database when no writers
  - Writers can access database when no readers
  - Only one thread manipulates state variables at a time
- Basic structure of a solution:
  - Reader()

```
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
```

Basic Readers/Writers Solution



- Correctness Constraints:
  - Readers can access database when no writers
  - Writers can access database when no readers
  - 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

Code for a Reader

Code for a Reader

```
Reader() {
  // First check self into system
  lock.Acquire();
```

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
  }
```

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();
```

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
  lock.release();             active!

  // Perform actual read-only access
  AccessDatabase(ReadOnly);
```

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
  lock.release();             active!

  // 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();
}
```

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() // Sleep on cond var
    WR--;
  }
  AR++;
  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();
}
```

Why is lock realeased here ?

Code for a Writer

Code for a Writer

```
Writer() {
  // First check self into system
  lock.Acquire();
```

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
  }
```

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();
```

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);
```

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();
}
```

Simulation of Readers/Writers solution

- Consider the following sequence of operators:
  - R1, R2, W1, R3

Simulation of Readers/Writers solution

- Consider the following sequence of operators:
  - R1, R2, W1, R3
- On entry, each reader checks the following:

```
while ((AW + WW) > 0) {  // Is it safe to read?
exist                    WR++;  // No. Writers

(&lock);                 okToRead.wait
                         // Sleep on cond var
waiting                  WR--;  // No longer

   }
AR++;                    // Now we are active!
```

Simulation of Readers/Writers solution

- Consider the following sequence of operators:
  - R1, R2, W1, R3
- On entry, each reader checks the following:

```
while ((AW + WW) > 0) {  // Is it safe to read?
exist                    WR++;  // No. Writers

(&lock);                 okToRead.wait
                         // Sleep on cond var
waiting                  WR--;  // No longer

   }
AR++;                    // Now we are active!
```

- First, R1 comes along:

  AR = 1, WR = 0, AW = 0, WW = 0

Simulation of Readers/Writers solution

- Consider the following sequence of operators:
  - R1, R2, W1, R3
- On entry, each reader checks the following:

```
while ((AW + WW) > 0) {  // Is it safe to read?
exist                    WR++;  // No. Writers

(&lock);                 okToRead.wait
                         // Sleep on cond var
waiting                  WR--;  // No longer

   }
AR++;                    // Now we are active!
```

- First, R1 comes along:

  AR = 1, WR = 0, AW = 0, WW = 0

- Next, R2 comes along:

  AR = 2, WR = 0, AW = 0, WW = 0

Simulation(2)

Simulation(2)

- Next, W1 comes along:

```
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++;
```

Simulation(2)

- Next, W1 comes along:

```
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++;
```

- Can't start because of readers, so go to sleep:

  AR = 2, WR = 0, AW = 0, WW = 1

Simulation(2)

- Next, W1 comes along:

```
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++;
```

- Can't start because of readers, so go to sleep:

  AR = 2, WR = 0, AW = 0, WW = 1

- Finally, R3 comes along:

  AR = 2, WR = 1, AW = 0, WW = 1

Simulation(2)

- Next, W1 comes along:

```
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++;
```

- Can't start because of readers, so go to sleep:

  AR = 2, WR = 0, AW = 0, WW = 1

- Finally, R3 comes along:

  AR = 2, WR = 1, AW = 0, WW = 1

- Now, say that R2 finishes before R1:

  AR = 1, WR = 1, AW = 0, WW = 1

Simulation(2)

- Next, W1 comes along:

```
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++;
```

- Can't start because of readers, so go to sleep:

  AR = 2, WR = 0, AW = 0, WW = 1

- Finally, R3 comes along:

  AR = 2, WR = 1, AW = 0, WW = 1

- Now, say that R2 finishes before R1:
  AR = 1, WR = 1, AW = 0, WW = 1

- Finally, last of first two readers (R1) finishes and wakes up writer:

Simulation(2)

- Next, W1 comes along:

```
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++;
```

- Can't start because of readers, so go to sleep:

  AR = 2, WR = 0, AW = 0, WW = 1

- Finally, R3 comes along:

  AR = 2, WR = 1, AW = 0, WW = 1

- Now, say that R2 finishes before R1:
  AR = 1, WR = 1, AW = 0, WW = 1

- Finally, last of first two readers (R1) finishes and wakes up writer:

```
if (AR == 0 && WW > 0)   // No other active readers
   okToWrite.signal();   // Wake up one writer
```

Simulation(3)

Simulation(3)

- When writer wakes up, get:

  AR = 0, WR = 1, AW = 1, WW = 0

Simulation(3)

- When writer wakes up, get:

  AR = 0, WR = 1, AW = 1, WW = 0

- Then, when writer finishes:

Simulation(3)

- When writer wakes up, get:

  AR = 0, WR = 1, AW = 1, WW = 0

- Then, when writer finishes:

```
while ((AW + WW) > 0) {  // Is it safe to read?
  WR++;                  // No. Writers exist
  okToRead.wait(&lock);  // Sleep on cond var
  WR--;                  // No longer waiting
}
```

Simulation(3)

- When writer wakes up, get:

  AR = 0, WR = 1, AW = 1, WW = 0

- Then, when writer finishes:

```
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!
```

  - Writer wakes up reader, so get:

    AR = 1, WR = 0, AW = 0, WW = 0

Simulation(3)

- When writer wakes up, get:

  AR = 0, WR = 1, AW = 1, WW = 0

- Then, when writer finishes:

```
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!
```

  - Writer wakes up reader, so get:

    AR = 1, WR = 0, AW = 0, WW = 0

- When writer completes, we are finished