Readers/Writers problem

Correctness Constraints:

- Readers can access the database when no writers
- Writers can access the database when no readers or writers
- Only one thread manipulates state variables at a time

Monitor-based Solution 1 (Giving writers a preference)

State variables (Protected by a 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 variables

Condition okToRead = NIL

Condition okToWrite = NIL

```
Reader() {
     // First check self into system
     acquire(&lock);
     while ((AW + WW) > 0) {        // Is it safe to read?
               WR++;           // No. Writers exist
               cond_wait(&okToRead,&lock);// Sleep on cond var
               WR--;           // No longer waiting
     }
     AR++;                 // Now we are active!
     release(&lock);         // why releasing the lock?
```

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

    // Now, check out of system
    acquire(&lock);
    AR--;               // No longer active
    if (AR == 0 && WW > 0)        // No other active readers
        cond_signal(&okToWrite);// Wake up one writer
    release(&lock);
}


Writer() {
    // First check self into system
    acquire(&lock);

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

        AW++;          // Now we are active!
    release(&lock);


    AccessDatabase(ReadWrite); // Perform actual write access

    // Now, check out of system
    acquire(&lock);
    AW--;               // No longer active
    if (WW > 0){        // Give priority to writers
        cond_signal(&okToWrite);// Wake up one writer
    } else if (WR > 0) {          // Otherwise, wake reader
        cond_broadcast(&okToRead); // Wake all readers
    }
    release(&lock);
}
```

Semaphore Solution 2: (giving readers a preference)

Use:

- A semaphore for mutual exclusion – mutex
- A semaphore for constraint for ordering between readers & writers - wlock

```
Reader() {
     // semaphore solution


     semaP(&mutex);       // reader enters to the CS

     AR++;

     if (AR == 1) {      // if first reader
               semaP(&wlock);      // get writer lock,
preventing writer
     }

     semaV(&mutex);  // allow other readers


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

          // Now, check out of system
     semaP(&mutex);
     AR--;
     if (AR == 0)   // if no other active readers
          semaV(&wlock)  // release writer lock
     semaV(&mutex);
}


Writer() {
     // Semaphore solution
     semaP(&wlock);


     AccessDatabase(ReadWrite); // Perform actual write access


     semaV(&wlock);
}
```