Concurrent Programming:
Synchronizing threads:
4. *The Readers-Writers Problem*

Readers-Writers Problem

- **Generalization of the Mutual Exclusion Problem**
  - Courtois et al., CACM, 1971
  - https://dl-acm-org.ezproxy.depaul.edu/doi/pdf/10.1145/362759.362813
- **Problem statement:**
  - *Writer* threads modify the object
  - *Reader* threads only read the object
  - Writers **must** have exclusive access to the object
  - Readers **do not need** exclusive access to the object:
    - Unlimited number of readers can read object concurrently!
- **Occurs frequently in real systems, e.g.,**
  - Online airline reservation system
  - Multithreaded caching Web proxy

Variants: Who gets unblocked first?

- ***First readers-writers problem* (favors readers)**
  - No reader should be kept waiting unless a writer has already been granted permission to use the object
  - A reader that arrives after a waiting writer *gets priority* over the writer

- ***Second readers-writers problem* (favors writers)**
  - Once a writer is ready to write, it performs its write as soon as possible
  - A reader that arrives after a writer must wait, even if the writer is also waiting

Variants: Who gets unblocked first?

- ***First readers-writers problem* (favors readers)**
  - No reader should be kept waiting unless a writer has already been granted permission to use the object
  - A reader that arrives after a waiting writer *gets priority* over the writer

- ***Second readers-writers problem* (favors writers)**
  - Once a writer is ready to write, it performs its write as soon as possible
  - A reader that arrives after a writer must wait, even if the writer is also waiting

- ***Starvation* (where a thread waits indefinitely) is possible in both cases**
  - *Third readers-writers problem* (FIFO): array of semaphores…

Solution to First Readers-Writers Problem

Readers:

```
int readcnt; /* Init = 0 */
sem_t mutex; /* Init = 1, for readcnt */
sem_t w;     /* Init = 1 */

void reader(void) {
    while (1) {
        P(&mutex);
        readcnt++;
        if (readcnt == 1) /* First in */
            P(&w);
        V(&mutex);

        /* Critical section */
        /* Multiple readers possible  */

        P(&mutex);
        readcnt--;
        if (readcnt == 0) /* Last out */
            V(&w);
        V(&mutex);
    }
}
```

Writers:

```
void writer(void)
{
    while (1) {
        P(&w);

        /* Critical section */
        /* Writing happens  */

        V(&w);
    }
}
```

rw1.c

Solution to First Readers-Writers Problem

Readers:

```
int readcnt; /* Init = 0 */
sem_t mutex; /* Init = 1, for readcnt */
sem_t w;     /* Init = 1 */

void reader(void) {
    while (1) {
        P(&mutex);
        readcnt++;
        if (readcnt == 1) /* First in */
            P(&w);
        V(&mutex);

        /* Critical section */
        /* Multiple readers possible  */

        P(&mutex);
        readcnt--;
        if (readcnt == 0) /* Last out */
            V(&w);
        V(&mutex);
    }
}
```

Writers:

```
void writer(void)
{
    while (1) {
        P(&w);

        /* Critical section */
        /* Writing happens  */

        V(&w);
    }
}
```

rw1.c

What about the *second* readers-writers problem? (See paper, 5 semaphores!)