CS 537 Notes, Section #7: Semaphore Example: Readers and Writers

---

Semaphore usage generally falls into two classes:

1. Uniform resource usage, simple first-in-first-out scheduling: use semaphores for everything. This is usually the case. Use one semaphore for each constraint in the system.
2. More complex patterns of resource usage: interaction between different users of a resource, or changing priorities: semaphores cannot capture the scheduling all by themselves. Must use *state variables* to record information about priorites, resource state. In this case, semaphores get used for two things:
    - One semaphore for mutual exclusion on the state variables.
    - One semaphore for each class of waiting; used just as a convenience to make a process wait. In the worst case, one semaphore per process.

    Whenever possible, cast problems into the first class. This usually can be done.

    Unfortunately, sometimes a resource is shared by different *classes* of users; that is, they use the resource in different ways. Potentially the different kinds of usage interact. For example, consider a shared database with readers and writers. It is safe for any number of readers to access the database simultaneously, but each writer must have exclusive access. Example: checking account (statement-generators are readers, tellers are writers).

    - Note that writers are actually readers too.
    - In this case, the constraints are too complicated to be solved single-handedly with semaphores.
    - Constraints:
        - Readers can only proceed if there are no active or waiting writers (use semaphore OKToRead).
        - Writers can only proceed if there are no active readers or writers (use semaphore OKToWrite).
        - Only one process may manipulate internal state variables at once (use semaphore Lock).
    - Scheduling: writers get preference.
    - State variables:
        - AR = number of active readers.
        - WR = number of waiting readers.
        - AW = number of active writers.
        - WW = number of waiting writers.

      AW is always 0 or 1. AR and AW may not both be non-zero.

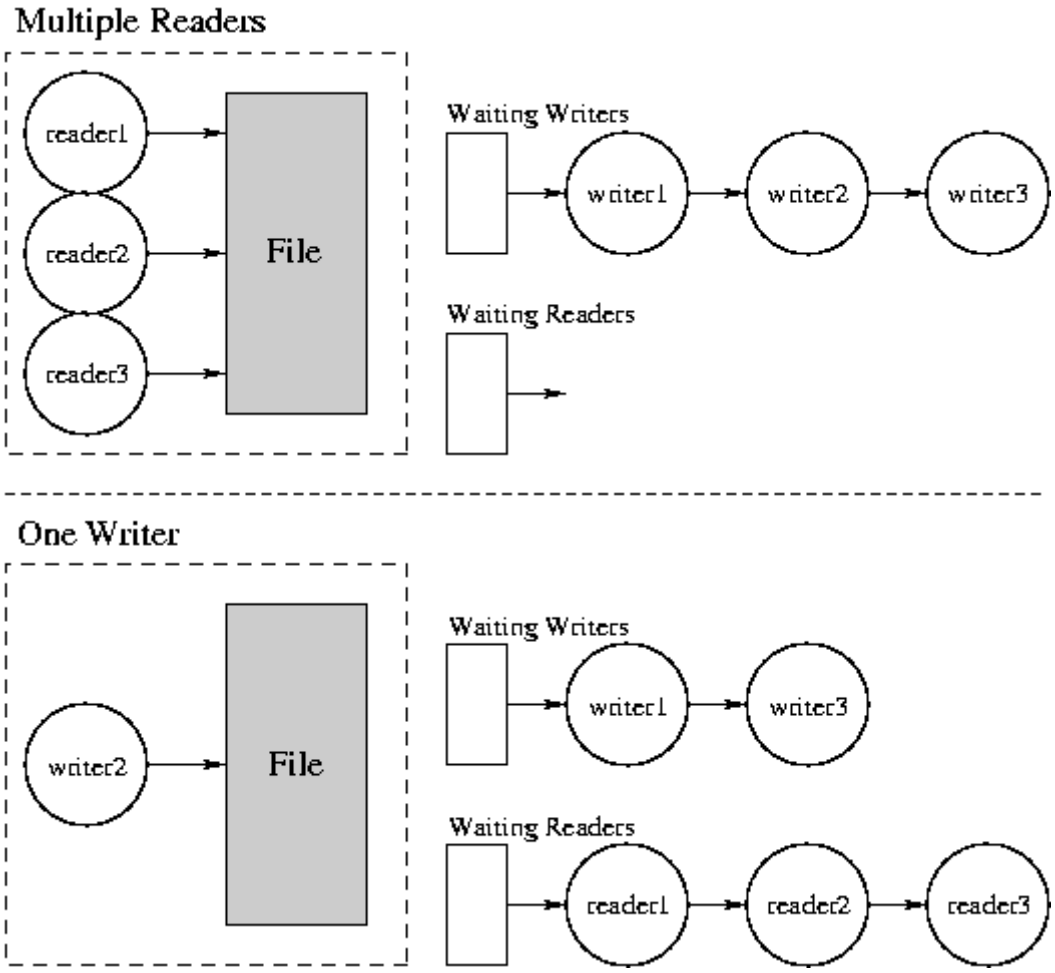
    - Initialization:
        - `semaphore OKToRead = new semaphore(0);`

- `semaphore OKToWrite = new semaphore(0);`
- `semaphore Lock = new semaphore(1);`
- `int AR = 0, WR = 0, AW = 0, WW = 0;`

Reader Process:	Writer Process:
`StartRead () { Lock.P(); if ((AW+WW) == 0) { OKToRead.V(); AR++; } else { WR++; } Lock.V(); OKToRead.P(); }`	`StartWrite () { Lock.P(); if ((AW+AR+WW) == 0) { OKToWrite.V(); AW++; } else { WW++; } Lock.V(); OKToWrite.P(); }`
`EndRead () { Lock.P(); AR--; if ((AR == 0) and (WW > 0)) { OKToWrite.V(); AW++; WW--; } Lock.V(); }`	`EndWrite () { Lock.P(); AW--; if (WW>0) { OKToWrite.V(); AW++; WW--; } else { while (WR>0) { OKToRead.V(); AR++; WR--; } } Lock.V(); }`
`main (); { StartRead(); // --read the necessary data-- EndRead(); }`	`main (); { StartWrite(); // --write the necessary data-- EndWrite(); }`

Examples:

- Reader enters and leaves system.
- Writer enters and leaves system.
- Two readers enter system.
- Writer enters system and waits.
- Reader enters system and waits.
- Readers leave system, writer continues.
- Writer leaves system, last reader continues and leaves.



Questions:

- In case of conflict between readers and writers, who gets priority?
- Is the WW necessary in the writer's first if?
- Can OKToRead ever get greater than 1? What about OKToWrite?
- Is the first writer to execute Lock.P() guaranteed to be the first writer to access the data?
