Readers–Writers Problem

Readers–Writers Problem

- Suppose that a database is to be shared among several concurrent processes.

Readers

- Those processes want only to read the database.

Writers

- Those processes want to update (that is, to read and write) the database.
- *If two readers access the shared data* simultaneously, no adverse effects will result.
- If a writer and some other process (either a reader or a writer) access the database simultaneously, adverse effects will result.

Readers–Writers Problem

- We require that the writers have exclusive access to the shared database while writing to the database.
- This synchronization problem is referred to as the **readers–writers problem.**

<u>The readers–writers problem has several variations,</u>

First readers–writers problem

- Requires that no reader be kept waiting unless a writer has already obtained permission to use the shared object.
- In other words, no reader should wait for other readers to finish simply because a writer is waiting.

***Second readers*–writers problem**

- Requires that, once a writer is ready, that writer perform its write as soon as possible.
- In other words, if a writer is waiting to access the object, no new readers may start reading.

- A solution to either problem may result in starvation. In the first case, writers may starve; in the second case, readers may starve.

Solution to the First Readers Writers Problem

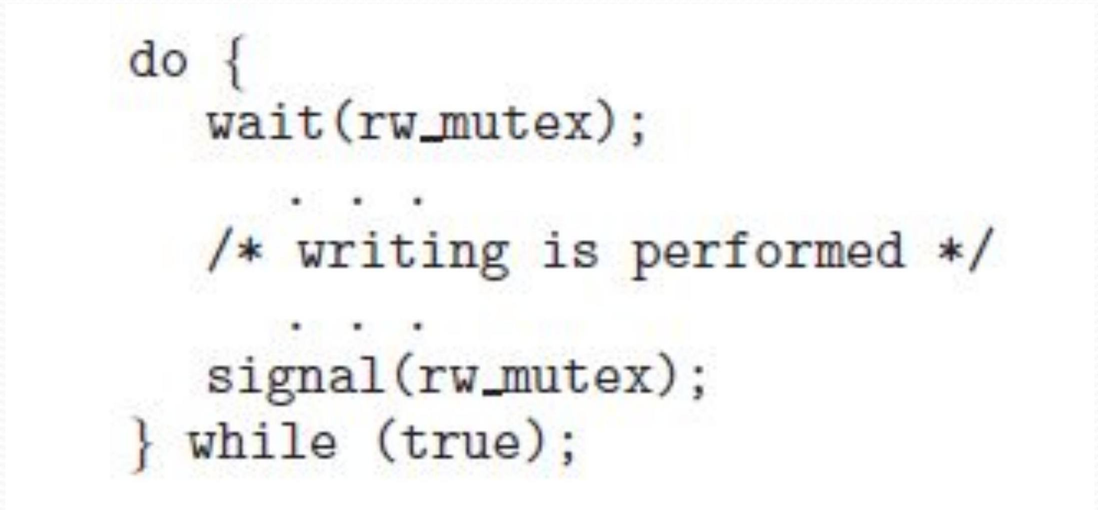
- Solution to the first readers–writers problem, the reader processes share the following data structures:

```
semaphore rw mutex = 1;
semaphore mutex = 1;
int read count = 0;
```

- The semaphores mutex and rw mutex are initialized to 1; read count is initialized to 0.
- The semaphore rw mutex is common to both reader and writer processes.
- The mutex semaphore is used to ensure mutual exclusion when the variable read count is updated.
- The read count variable keeps track of how many processes are currently reading the object.

Solution to the First Readers Writers Problem

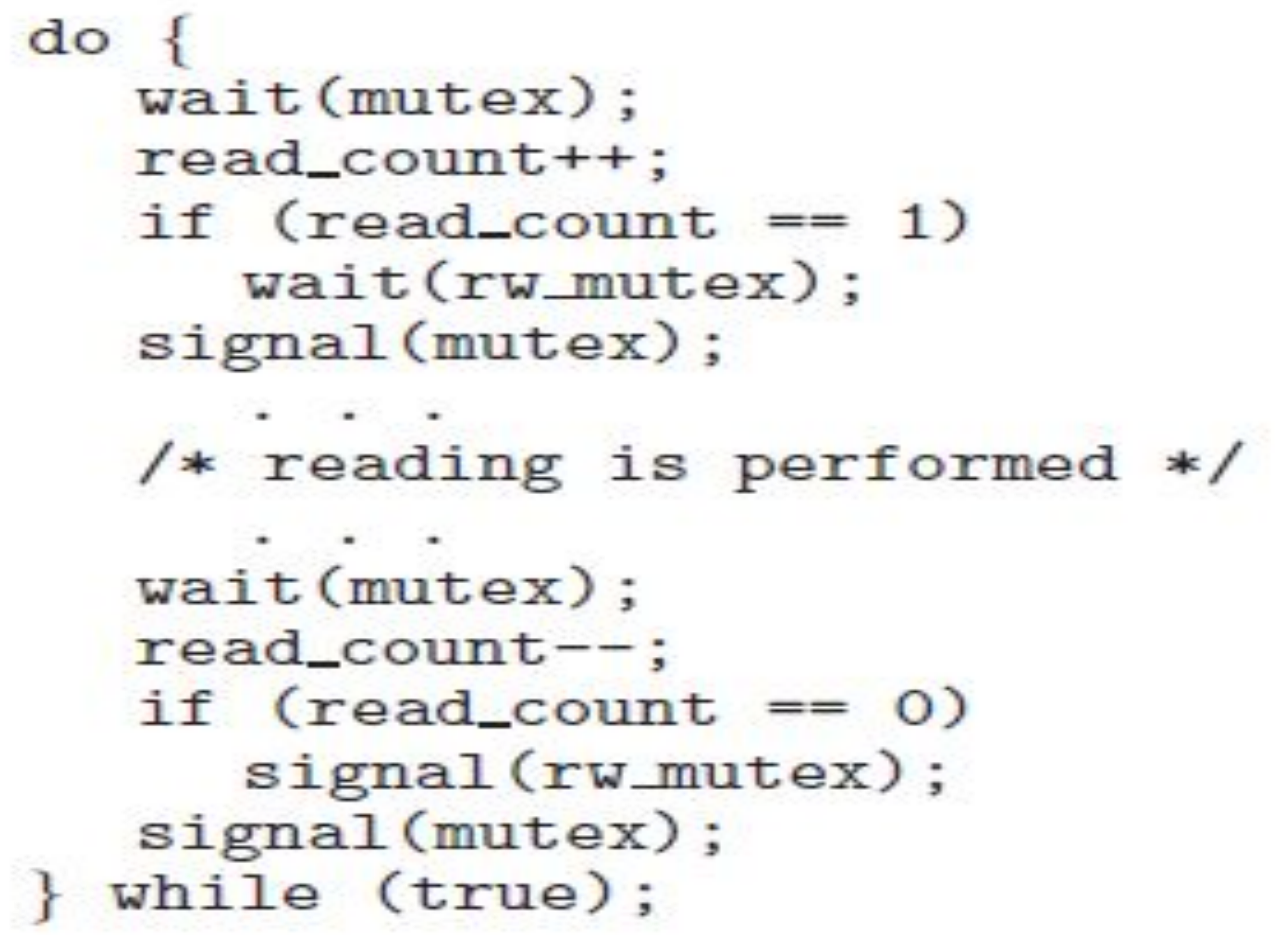
- The semaphore rw mutex functions as a mutual exclusion semaphore for the writers.
- It is also used by the first or last reader that enters or exits the critical section.
- It is not used by readers who enter or exit while other readers are in their critical sections



```
do {
   wait(rw_mutex);
      . . .
   /* writing is performed */
      . . .
   signal(rw_mutex);
} while (true);
```

The structure of a writer process.

Solution to the First Readers Writers Problem



```
do {
    wait(mutex);
    read_count++;
    if (read_count == 1)
       wait(rw_mutex);
    signal(mutex);
       . . .
    /* reading is performed */
       . . .
    wait(mutex);
    read_count--;
    if (read_count == 0)
       signal(rw_mutex);
    signal(mutex);
} while (true);
```

The structure of a reader process.

Solution to the First Readers Writers Problem

- If a writer is in the critical section and *n readers are waiting, then one reader is queued on rw mutex, and n − 1 readers are queued on mutex.*
- *When a writer executes* signal(rw mutex),we may resume the execution of either the waiting readers or a single waiting writer. The selection is made by the scheduler.