Reykjavik University

T-419-CADP - Assignment 1

Readers and Writers

Freyr Elí Sveinbjörnsson
freyr23@ru.is

Georg Ingi Einarsson
georg23@ru.is

Óliver Máni Samúelsson
olivers23@ru.is

Instructor: Marcel Kyas

January 30, 2026

1 Readers and Writers - Solution with Writer Preference

We consider the solution by Courtois, Heymans, and Parnas (1971), which implements writer priority. The shared variables are:

nr = number of active readers, nw = number of writers waiting or writing.

All semaphores (m1, m2, m3, writer, and reader) are initialized to 1, and $nr = nw = 0$ initially.

The code for this solution can be seen at the end of the report.

1.1 Role of the Semaphores

- `m1`: A mutex protecting the reader counter `nr`. It ensures that updates to `nr` and the first/last reader logic are performed atomically. It encapsulates a check that ensures that the writers can't write while ongoing readers are reading.
- `m2`: A mutex protecting the writer counter `nw`. It ensures atomic updates to `nw` and correct first/last writer behavior, so that only one writer can write at a time. It also gives writers priority (lines 34-39), if a writer wants to write, we block new readers from attempting to read by decrementing the reader semaphore.
- `m3`: Ensures that at most one reader at a time attempts to pass the `reader` semaphore. This avoids race conditions where a reader could be awakened instead of a writer.
- `reader`: A gate semaphore that writers use to block the arrival of new readers. When held by writers, no new reader may begin.
- `writer`: Ensures exclusive access to the database. It is held either by exactly one writer or collectively by all readers. If `writer` is held, no other writer and no reader may access the database.

1.2 Assertions and Invariants

Global Invariants The following properties hold at all times:

- $nr \geq 0$ and $nw \geq 0$.
- If $nr > 0$, then the semaphore `writer` is held.
- If a writer is in the database, then $nr = 0$.
- At most one writer can hold `writer` at any time.

Reader Entry When a reader increments `nr`:

- If $nr = 1$, the reader performs `writer.P()`, preventing writers from entering the database.
- Subsequent readers may enter without blocking on `writer`.

Reader Critical Section While a reader is accessing the database:

$$nr > 0 \quad \text{and} \quad \texttt{writer} \text{ is held.}$$

Thus, no writer can access the database concurrently.

Reader Exit When a reader decrements `nr`:

- If $nr = 0$, the reader performs `writer.V()`, allowing a writer to enter.

Writer Entry When a writer increments `nw`:

- If $nw = 1$, the writer performs `reader.P()`, blocking all new readers from entering.

Writer Critical Section A writer must acquire `writer` before accessing the database. While holding it:

$$nr = 0 \quad \text{and} \quad \text{no other writer holds } \texttt{writer}.$$

Hence, writers have exclusive access to the database.

Writer Exit When a writer decrements `nw`:

- If $nw = 0$, the writer performs `reader.V()`, allowing readers to enter again.

1.3 Mutual Exclusion for Writers

The semaphore `writer` guarantees mutual exclusion:

- Only one writer can hold `writer` at a time.
- Readers must acquire `writer` (via the first reader), so no reader can access the database while a writer holds it.

Therefore, if a writer is accessing the database, all other writers and all readers are excluded.

1.4 Writer Preference Argument

This solution prefers writers because:

- The first arriving writer immediately locks `reader`, preventing new readers from entering.
- Existing readers are allowed to finish, but no new readers may start.
- Once the last reader exits, a writer can immediately acquire `writer` and access the database.

Thus, once a writer arrives, it is guaranteed to eventually access the database before any newly arriving readers, which establishes writer priority.

```
1  func Reader() {
2      for {
3          m3.P()
4              reader.P()
5                  m1.P()
6                      nr++
7                      if nr == 1 {
8                          writer.P()
9                      }
10                 m1.V()
11             reader.V()
12         m3.V()
13         // Access the data base
14         m1.P()
15             nr--
16             if nr == 0 {
17                 writer.V()
18             }
19         m1.V()
20     }
21 }
22
23 func Writer() {
24     for {
25         m2.P()
26             nw++
27             if nw == 1 {
28                 reader.P()
29             }
30         m2.V()
31         writer.P()
32         // Read and write the data base
33         writer.V()
34         m2.P()
35             nw--
36             if nw == 0 {
37                 reader.V()
38             }
39         m2.V()
40     }
41 }
```


Figure 1: Reader and Writer code