# Readers and Writers

Demonstrated by Ýmir Þórleifsson & Bjartur Þórhallsson

Role of semaphores

* **m1**

Since we are accessing a shared variable nr (number of readers) we must block all other readers from accessing that variable at the same time. This is done with the m1 semaphore.

* **m2**

The m2 semaphore takes care of the same problem as m1 only for the nw (number of writers) variable.

* **m3**

The purpose of this semaphore is to only allow a single reader to be blocked on the read semaphore. When either a reader or a writer calls V(read) the reader will access the next semaphore, m1, which protects the nr variable. This way the nr variable is always protected and we keep the writer preference throughout the algorithm.

* **1read**

The reader semaphore makes sure that specific functions in the Reader() can only be accessed by a single process at a time. The function that only a single Reader can execute is where the nr counter is incremented and a check is made if the current reader process is the first one, if this is true, we block the writer semaphore. We don´t want many processes to execute this line concurrently, thus we need the reader semaphore.

Writers also use the read semaphore to block readers from entering. When the first reader wants to enter, it blocks other readers from entering the database.

* **write**

The usage of the write semaphore is simple. Since we only want a single writer to access the database at a time, we need to surround the database access function with the writer semaphore, decremented when the writer is about to access it, and incremented when the process leaves.

The reader process also performs actions on the writer semaphore, when there is a reader process about to enter the database, and there are currently no other readers in the database, the reader process about to enter blocks other writers from entering, until the last reader exits and lets writers enter.

Writer preference

The solution obviously has a writer preference where a writer will block all incoming readers until there are no writers “wanting” to enter. There can also only be one writer in the database at a time, but this makes sense. However, whenever there are readers waiting on a writer to finish and another writer process want to enter the database, the new writer process has priority over the waiting readers.

The new writer simply increments the nw variable and continues until he wants to enter the database and decrements the writers semaphore which he will get blocked on. When the first writer exits the database, he increments the writer semaphore, letting the second writer into the database.

The first writer decrements nw by one and checks if he is the last writer to enter the process. This is the part that excludes reader processes from ever entering, unless there are currently no writers wanting to enter or currently in the database. The writer only lets writers in if it is the last writer to go through the database. This allows for starvation for readers, i.e., if we have a continuous flow of writers wanting access to the database, the readers that are waiting will never get access.

Mutual exclusion amongst writers

The solution allows for mutual exclusion amongst writers.

When a writer wants to enter the database, he must first increment the shared nw variable. He does this by decrementing m2, which is the semaphore that “protects” the nw variable. If he is not blocked on the m2 decrement, he increments nw and checks if he is the first writer to enter, if he is, he blocks readers from entering. Then he continues and tries to decrement the writer semaphore in order to access the database. The writer semaphore makes sure that only a single writer process is in the database at a time.

After accessing the database, he signals increments the writer semaphore to let other writers into the database, then he wants to decrement nw, he can only do this if there is no other writer process accessing the nw variable. This is due to the m2 semaphore. After decrementing the nw, he checks if he is the last writer to exit the database, if this is true, he increments the read semaphore in order to give readers access.

Writers are always able to access the database, one after the other. They can skip over the readers waiting for access, but solution uses semaphores perfectly to provide mutual exclusion to the database and nw variable.