Concurrent and Distributed Systems
Concurrent Objects

Mitre Flavia Antonia

December 21, 2019

Third Year
Group: C.E.N. 3.1
Lecturer: Costin Bădică
Teaching assistant: Cristinel Ungureanu

1 Problem statement

1.1 Second problem

Implement the readers-writers problem using semaphores as follows:
- More readers can read at the same time the same shared resource as long as there is no writer with writing access to the same shared resource
- A writer will get exclusive access to the shared resource (no other writer and/or reader can have access at the same time to the same shared resource)
- There is no starvation for readers or writers

2 Implementation

2.1 Second problem

For solving the second problem with **semaphores** I've created the class:

- **ReadersWriters**

  Two fair **semaphores** with only one permit are declared here: one for the **reader** and one for the **writer** and the variable **CountReaders** used to count the number of readers at a certain time.

  This class contains two static classes: **Reader** and **Writer** and the **main** of the program.

- **Reader**

  This class implements **runnable** (overrides the run method from threads).
  The **reader** tries to **acquire** the semaphore, the **reader count** is **incremented**.
  If the reader count is equal to **one** (at least one reader reads at the moment), the **writer** is **acquired** (the writer has exclusive access to the shared source).
  The **reader** is **released**, so more readers can read at the same time.
  The message for **reading** is displayed.
  The reader has access to the source for some **time**.
  The message for **finishing reading** is displayed.
  The **reader** tries to **acquire** the semaphore, the **reader count** is **decremented**.
  If the reader count is equal to **zero** (nobody reads at the moment), the **writer** is **released** (the writer can have access to the shared source).
  The **reader** is **released**.

- **Writer**

  This class implements **runnable** (overrides the run method from threads).
  The **writer** tries to **acquire** the semaphore.
  The message for **writing** is displayed.
  The writer has access to the source for some **time**.
  The message for **finishing writing** is displayed.
  The **writer** is **released**.

Pseudocode

```
1. Readers count ← 0
2. Semaphore reader
3. Semaphore writer
```

Reader:

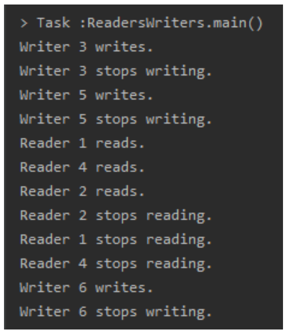
```
4. wait(reader)
5. CountReaders ← CountReaders + 1
6. **if** CountReaders = 1 **then**
7. wait(writer)
8. signal(reader)
9. read
10. wait(reader)
11. CountReaders ← CountReaders - 1
12. **if** CountReaders = 0 **then**
13. signal(writer)
14. signal(reader)
```

Writer:

```
15. wait(writer)
16. write
17. signal(writer)
```

3 Experimental Data

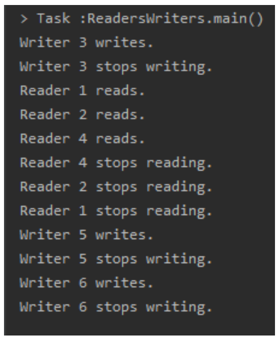
3.1 Second problem



```
> Task :ReadersWriters.main()
Writer 3 writes.
Writer 3 stops writing.
Writer 5 writes.
Writer 5 stops writing.
Reader 1 reads.
Reader 4 reads.
Reader 2 reads.
Reader 2 stops reading.
Reader 1 stops reading.
Reader 4 stops reading.
Writer 6 writes.
Writer 6 stops writing.
```

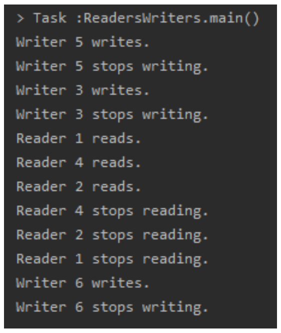
Figure 1: First output

Writer 3 writes and stops writing after some time.
Writer 5 writes and stops writing after some time.
Reader 1 starts reading.
Reader 4 starts reading.
Reader 2 starts reading.
Reader 2 stops reading.
Reader 1 stops reading.
Reader 4 stops reading.
Writer 6 writes and stops writing after some time.
More readers can have access to the shared source at the same time.
The writers have exclusive access to the shared source.
All the readers and all the writers manage to do their action (no starvation).



```
> Task :ReadersWriters.main()
Writer 3 writes.
Writer 3 stops writing.
Reader 1 reads.
Reader 2 reads.
Reader 4 reads.
Reader 4 stops reading.
Reader 2 stops reading.
Reader 1 stops reading.
Writer 5 writes.
Writer 5 stops writing.
Writer 6 writes.
Writer 6 stops writing.
```

Figure 2: Second output
Writer 3 writes and stops writing after some time.
Writer 5 writes and stops writing after some time.
Reader 1 starts reading.
Reader 4 starts reading.
Reader 2 starts reading.
Reader 2 stops reading.
Reader 1 stops reading.
Reader 4 stops reading.
Writer 6 writes and stops writing after some time.



```
> Task :ReadersWriters.main()
Writer 5 writes.
Writer 5 stops writing.
Writer 3 writes.
Writer 3 stops writing.
Reader 1 reads.
Reader 4 reads.
Reader 2 reads.
Reader 4 stops reading.
Reader 2 stops reading.
Reader 1 stops reading.
Writer 6 writes.
Writer 6 stops writing.
```

Figure 3: Third output
Writer 5 writes and stops writing after some time.
Writer 3 writes and stops writing after some time.
Reader 1 starts reading.
Reader 4 starts reading.
Reader 2 starts reading.
Reader 4 stops reading.
Reader 2 stops reading.
Reader 1 stops reading.
Writer 6 writes and stops writing after some time.

4 Conclusions

We can consider a situation where many people share the same file.
If a person tries to edit the file, no one else should be reading or writing at the same time, so changes will not be visible to them.
If someone is reading the file, others may read it at the same time.

The readers-writers problem has the following constraints:

-One set of data is shared among a number of processes
-Once a writer is ready, it performs its write. Only one writer may write at a time
-If a process is writing, no other process can read it
-If at least one reader is reading, no other process can write
-Readers may not write and only read

Using **semaphores** and a **readers counter** that is incremented/decremented guarantee that all the above constraints are met.