Seminar Report: Muty

Albert Bausili, Noa Yu Ventura, Pol Verdura

July 25, 2023

Upload your report in PDF format.
Use this LaTeX template to format the report.

Try to answer all the open questions in the assignment. When asked to do so, perform experiments to support your answers.

1 Open questions

a) What unwanted halting situation can occur when contention increases?
The situation that can occur when using lock1 is called Deadlock and consists of a sequence of requests to enter the critical section that cannot be fulfilled. Both processes are trying to enter the critical section resulting in both of them remaining halted as neither of them is giving the "OK" to the other one, so they both remain to wait for that last okay that never arrives until they give up.

b) Indicate two situations which can lead to a process *withdrawal* (i.e., a process gets tired of waiting to be granted access to the critical section). Assume that processes do not fail.
- **Situation 1**: There might be a deadlock, so after some time passes the worker gets tired of waiting and tells its lock to stop by sending a release message.
- **Situation 2**: If the time spent in the critical zone is longer that the wait time, at the end, the workers that are waiting to enter the critical zone will get tired and will withdraw the request.

c) Justify how your code guarantees that only one worker is in the critical section at any time.
We can only enter the critical section when our list of locks that did not send an `ok` message is empty, and since we do not send any `ok` message to requests with lower priority than yourself when you want to enter the critical section, we are guaranteed to be alone in it.

d) Do the situations described in the previous question b) still lead to a process *withdrawal* in this lock implementation?
- **Situation 1**: No, they do not, because in this case there are no deadlocks. Implementing priorities avoids deadlocks, since there will always be a lock that can enter the critical section before another lock.

- **Situation 2**: Yes, because the critical time and waiting time is still the same. Plus priorities create an even more noticeable starvation.

e) What is the main drawback of this lock implementation?
We have starvation. A worker or a group of workers that want to constantly enter the critical section with a higher priority than another lock. The starved lock constantly receives requests from locks with higher priority, so in the end it sends an `ok` message to all of them but it is incapable of entering the critical section.

f) Justify if this lock implementation requires sending an additional `request` message (as well as the `ok` message) when a process in the `waiting` state receives a `request` message with the same logical time from another process with higher priority.
Since they have higher priority, yes we need to send the `ok` message. But the additional request message is not needed, since your main request has the same time as this higher priority process, which means that the next one is someone with the same logical time and higher priority than you, or yourself. You are taken into account by everyone because of the logical time in your main request.

g) Do the situations described in the previous question b) still lead to a process *withdrawal* in this lock implementation?
- **Situation 1**: No, because there are no deadlocks thanks to the same reason as in question d). Now we always have a lock that enters before another one, so there will always be someone that enters the critical section before than others (in this case not only priorities, but also a timestamp), which does not lead to a deadlock.
- **Situation 2**: Yes, because the critical time and waiting time is still the same, but at least it is more balanced than in the other strategies.

h) Note that the workers are not involved in the Lamport's clock. According to this, would it be possible that a worker is given access to a critical section prior to another worker that issued a request to its lock instance logically before (assuming happened-before order)? (Note that workers may send messages to one another independently of the mutual-exclusion protocol).
It could happen if the worker and its lock had such a high latency that the lock processed the OK message and send it (as it have not received any request from its worker) and after that received the request from its worker that came later than the OK message. If both are running in the same node this is highly improbable, but this could be an scenario if they were distributed in different nodes.

2 Personal opinion

Give your opinion of the seminar assignment, indicating whether it should be included in next year's course or not.

- **Noa:** I liked this assignment, it is very interesting to know how exclusion zones work in a distributed system, specially after coursing PAR, because the

solutions to the same problems are very different when you have one or multiple devices. I would include it in the next year's course, not extremely difficult but also not extremely easy. The code is also easy to understand, though having comments in the code would be more helpful, since we had to read some parts a lot of times to understand what it was doing.

- **Albert:** I found this assignment very interesting, as we've been able to see and feel the problems of a poor (or inexistent) organitzation of a shared resource can impact the whole system deeming it unusable. I think that the difficulty is acceptable and I would include this lab for the next year.

- **Pol:** I feel there's a big difficulty difference between the last lab and this one, it might be that there are no comments and you spend more time trying to understand the code than doing the other locks.