UNIVERSITÄT SIEGEN

Fakultät IV
Betriebssysteme und verteilte Systeme
Prof. Dr. rer. nat. Roland Wismüller

Excercise Sheet 9
Solution

Lecture Distributed Systems
Winter Term 2024/25

Exercise 1: Algorithm of Ricart and Agrawala

Suppose a process denies permission and then crashes. The requesting process believes it is still alive, but the permission never arrives. A way out is to cause the requester not only to block, but to *sleep* for a certain amount of time, after which he queries all processes that have denied permission to check if they are still running.

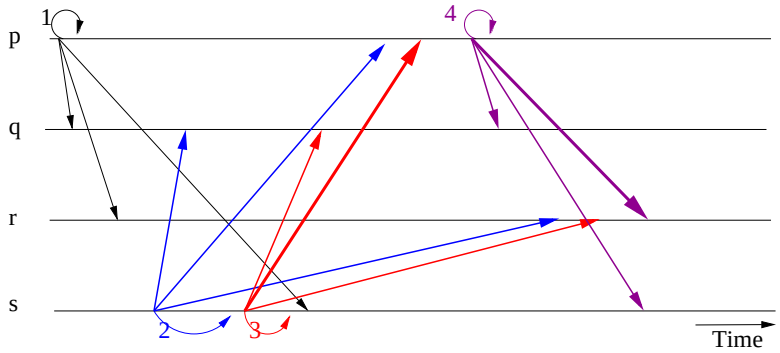
Exercise 2: Deadlocks - Ricart and Agrawala

That depends on the fundamental rules. If the processes enter into critical sections strictly sequentially, i.e. a process in a critical section must not attempt to enter into another critical section, there is no way it can block while holding a resource (i.e. a critical section) that another process needs. The system is deadlock-free. On the other hand, if process 0 can enter critical section A and then attempt to enter critical section B, a deadlock may occur if another process attempts to enter the critical sections in reverse order. The algorithm of Ricart and Agrawala itself does not contribute to the deadlock because each critical section is treated independently of the others.

Exercise 3: Programming: Ricart/Agrawala (Mandatory exercise for 6 CP, submit via moodle!)

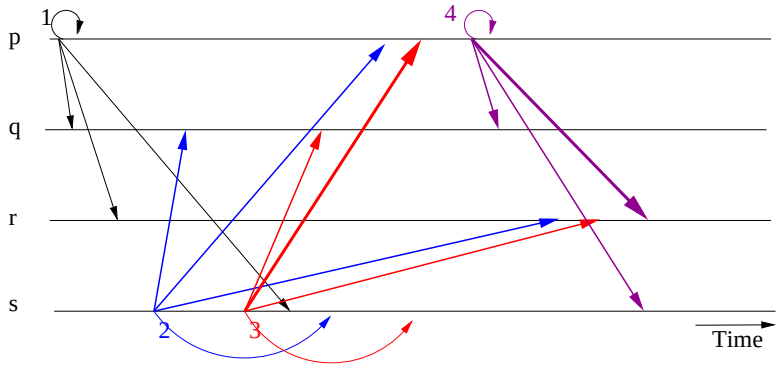
Exercise 4: Multicast Order Guarantees

a) With a causally sorted multicast, message 3 must arrive in all processes (especially process p) after message 2 (since 3 causally depends on 2). Message 4 must also arrive in all processes (especially process r) after message 3.



However, this does not guarantee total order (for example, process p: sequence 1,2,3,4; process s: sequence 2,3,1,4).

A total order would be provided if all messages from all processes were delivered in the same order.



b) For the user to receive all publications, a **reliable** multicast is required.

A **FIFO order** is required so that a user's posts, e.g. A. Bauer, are received everywhere in the same order. Users can then consistently discuss A. Bauer's "second posting". Also, the second post might refer to the first one.

A **causal order** is also needed because the messages whose topics begin with *Re:* should appear after the messages they refer to. Otherwise, delaying a message could cause the message *Re: RPC Principle* to appear before the original message *RPC Principle*.

If the multicast delivery were totally ordered, the numbering in the left column would be consistent between users. However, this is not absolutely necessary.

In practice, the USENET system, which emulates such a bulletin board, implements neither a causal nor a total order. The communication effort required for this would far outweigh its advantages.

Exercise 5: Totally Ordered Multicast

A second approach is to send the message immediately via multicast, but to postpone delivery until the sequencer has multicast a sequence number for it. This happens after the message is received by the sequencer.

A third approach is to first request a sequence number from the sequencer and then multicast the message.

The first approach (to send the operations to the sequencer) involves sending a point-to-point message with the operation and a multicast message.

The second approach requires two multicast messages: one with the operation and one with the sequence number.

The third approach costs a point-to-point message with the sequence number followed by a multicast message containing the operation.

Exercise 6: Causally Ordered Multicast (Mandatory exercise for 6 CP, submit via moodle!)