Challenge

It was shown in the lectures that access to a shared resource by a group of peer processes in a distributed mutual exclusion fashion may be achieved through the application of Maekawa's voting algorithm. This algorithm has the advantage, *vis a vis*, the Ricart-Agrawala's total order algorithm, of dramatically reducing the number of exchanged messages when the number of involved processes is very large.

However, the algorithm is incorrect in the original formulation. There are instances that, due to the prevailing racing conditions, deadlock may occur.

Assume, as in the example discussed in the lectures, that three processes p_0, p_1, and p_2, with $V_0 = \{p_0, p_1\}$, $V_1 = \{p_1, p_2\}$ and $V_2 = \{p_2, p_0\}$, try to access the same shared object about the same time. It may happen that, due to the times of messages arrival, p_0 votes for its own access, p_1 votes for its own access and p_2 votes for its own access. Thus, all processes receive a first permission, but never a second and deadlock ensures!

A possible solution, as proposed by Saunders, is to *total order* the access requests. Another one is to deny the circular waiting condition by *ordering access requests by the process id*.

Following that second line of thought, adapt the state diagram and the algorithm sketch presented in the lecture notes so that deadlock is prevented.

GRADING

- +1 unit on the course theoretical mark if completed with success.

DELIVERABLE

- a pdf file, named `Maekawa<name+nMec>.pdf`, with your answer.

DEADLINE

- June, 15, at midnight.