Exercises for today:

• 14.1, 14.3-14.7, 14.9-14.10 (chapter Mutual exclusion)
• 18.1, 18.7 (chapter Self-Stabilization)
Mutual exclusion algorithms should satisfy two properties:

- **Mutual exclusion**: in every configuration, at most one process is privileged.
- **Starvation-freeness**: if a process $p$ tries to enter its critical section, and no process remains privileged forever, then $p$ will eventually enter its critical section.
Exercise 14.1

Say for both mutual exclusion and starvation-freeness whether this is a safety or liveness property.
Reminder: Ricart – Agrawala algorithm + Carvalho – Roucairol optimization

Ricart – Agrawala:

• When $p_i$ wants to enter the critical section, it sends $(\text{request}, \text{ts}_i, i)$ to all processes
  • $\text{ts}_i$ – $p_i$’s logical clock
  • $i$ – $p_i$’s id
• When $p_j$ receives this request, it sends permission to $p_i$ as soon as:
  • $p_j$ isn’t privileged, and
  • $p_j$ doesn’t have a pending request with time stamp $\text{ts}_j$, where $(\text{ts}_j, j) < (\text{ts}_i, i)$ (lexicographical order)
• $p_i$ enters its critical section when it has received permission from all other processes.
• When $p_i$ exits its critical section, it sends permission to all pending requests

Carvalho – Roucairol optimization:

After a process $q$ has exited its critical section and wants to enter it again, $q$ only needs to send requests to the set $S$ of processes that $q$ has sent permission to since this exit.
Show that if processes could apply the Carvalho-Roucairol optimization from the start (instead of after a first entry of their critical section), then the resulting mutual exclusion algorithm would be incorrect.
The logical clock values in the Ricart-Agrawala algorithm are unbounded. Adapt the algorithm such that the range of these values becomes finite (using modulo arithmetic).
Reminder: Raymond’s algorithm

- Processes are organized in undirected network with sink tree.
- The root holds the token and is privileged.
- Each process maintains a FIFO queue, which can contain id’s of its children, and its own id. Initially, this queue is empty.

Queue maintenance:
- When a non-root wants to enter its critical section, it adds its id to its own queue.
- When a non-root gets a new head at its (non-empty) queue, it asks its parent for the token.
- When a process receives a request for the token from a child, it adds this child to its queue.
When the root exits its critical section (and its queue is non-empty):

- it sends the token to the process $q$ at the head of its queue,
- makes $q$ its parent,
- and removes $q$ from the head of its queue.

Let $p$ get the token from its parent, with $q$ at the head of its queue:

- If $q \neq p$, then $p$ sends the token to $q$, and makes $q$ its parent.
- If $q = p$, then $p$ becomes the root (i.e., it has no parent, and is privileged).

In both cases, $p$ removes $q$ from the head of its queue.
Run Raymond’s algorithm on the network from example 14.3. Initially, process $p_3$ holds the token, and all buffers are empty. Give a computation (including all messages) in which first $p_8$, then $p_2$, and finally $p_5$ requests the token, but they receive the token in the opposite order.
Exercise 14.6

Argue that in Raymond’s algorithm, each request to enter a critical section gives rise to at most $2D$ messages.
Exercise 14.7

Explain in detail why Raymond’s algorithm is starvation-free.
Reminder: Agrawal-El Abbadi algorithm

- N processes organized as a binary tree, $N = 2^{k+1} - 1$, k is the tree depth
- Processes need a **quorum** to enter the critical section.
- A quorum consists of:
  - all processes on a path from the root to a leaf, or
  - all processes on two paths from each child of $p$ to some leaf, if $p$ has crashed

- A process $p$ that wants to enter its critical section:
  - Places the root in its queue and tries to get permission of it
  - $p$ repeatedly tries to get permission from the head $r$ of its queue.
  - If successful, $r$ is removed from $p$’s queue.
  - If $r$ is a non-leaf, one of $r$’s children is appended to $p$’s queue.
  - If non-leaf $r$ has crashed, it is removed from $p$’s queue, and both of $r$’s children are appended at the end of the queue in fixed order
  - If leaf $r$ has crashed, $p$ aborts its attempt to become privileged.

When $p$’s queue becomes empty, it enters its critical section.
In the Agrawal-El Abbadi algorithm, what is the minimum and the maximum size of a quorum (in terms of $N$)?
Exercise 14.10

Prove that for each pair of quorums $Q$ and $Q'$ in the Agrawal-El Abbadi algorithm, $Q \subseteq Q'$ implies $Q = Q'$. 
A distributed algorithm is self-stabilizing if it will always end up in a correct configuration.
Reminder: Dijkstra’s token ring for mutual exclusion

- Processes $p_0, \ldots, p_{N-1}$ form a directed ring.
- Each $p_i$ holds a value $x_i \in \{0, \ldots, K - 1\}$ with $K \geq N$:
  - $p_i$ for $i = 1, \ldots, N - 1$ is privileged if $x_i \neq x_{i-1}$.
  - $p_0$ is privileged if $x_0 = x_{N-1}$.

- Each privileged process is allowed to change its value, causing the loss of its privilege:
  - $x_i \leftarrow x_{i-1}$ when $x_i \neq x_{i-1}$, for $i = 1, \ldots, N - 1$
  - $x_0 \leftarrow (x_0 + 1) \mod K$ when $x0 = x_0 = x_{N-1}$

- If $K \geq N$, then Dijkstra’s token ring self-stabilizes. That is, each execution eventually satisfies mutual exclusion and is starvation-free.
Exercise 18.1

• Give a computation of Dijkstra’s token ring with $N = K = 4$ that takes as long as possible before it reaches a correct configuration.
Reminder: Afek-Kutten-Yung self-stabilizing spanning tree algorithm

• Builds a spanning tree on undirected network. The process with highest id should be the root.

• Each process $p$ maintains the following variables:
  - $parent_p$ - its parent in the spanning tree
  - $root_p$ - the root of the spanning tree
  - $dist_p$ - its distance from the root via the spanning tree

• A non-root $p$ declares itself root ($parent_p \leftarrow \bot$, $root_p \leftarrow p$, $dist_p \leftarrow 0$) if it detects an inconsistency in its local variables or with the local variables of its parent:
  • $root_p \leq p$; or
  • $parent_p = \bot$; or
  • $parent_p \neq \bot$, and $parent_p$ is not a neighbor of $p$; or $root_p \neq root_{parent_p}$ or $dist_p \neq dist_{parent_p} + 1$
• Describe one possible computation of the Afek-Kutten-Yung algorithm on the network from exercise 18.3.