Seminar 3 – Messages & Sychronization

Each group will implement a small distributed system with nodes running on **different*** physical machines that solves the respective described problems.

Decompose the problems and use UDP to exchange messages (send/receive) between nodes. Failures do not need to be handled explicitly, as there will be another seminar that focuses on failure handling! Think about different ways of organizing the communication in your system and make an appropriate choice.

The discussion of the group's solution in the lab should include a demo (showing that the system works) and a presentation (explaining how and why it works) that also includes a sequence diagram showcasing a typical interaction. Think about how to do that demonstration in a nice way!

* It is okay to have multiple nodes on a single machine, but not all of them!

Group 1
Cigarette Smokers problem

Around a table a group of people is gathered, three smokers and one dealer. In order to facilitate their addiction, the smokers require 3 ingredients – paper, tobbaco & a match – to assemble a cigarette and consume it. The dealer has infinite amouts of all ingredients available. Each smoker has an infinite amount of one different ingredient available – one smoker has infinite paper, one has infinite matches, one has infinite tobacco.

The people are repeating the following steps over and over:

1. The dealer puts *two different, randomly selected* ingredients on the table using twice a `putIngredient`(i) method. The dealer will not communicate directly with the smokers at this stage.
2. The smoker who has the missing ingredient takes the two ingredients from the table (invoking a method `takeIngredient`(i) twice), uses them to make a cigarette and smokes it (which takes a certain amount of time). Then the smoker asks the dealer to continue.

The table should be implemented as a node itself, not only as a resource and provide the functions `putIngredient`(i), `takeIngredient`(i), and `checkIngredient`(i) with i = {1,2}.

Implement and explain a system that synchronizes the nodes according to the described problem.

Group 2
Roller Coaster problem

There is a roller coaster with n wagons and m pleasure-seeking passengers. Each passenger wants to ride the roller coaster repeatedly. Each wagon has seats for c passengers, $c<m$. A wagon only departs when all seats are occupied. After completing the coaster (which takes a certain amount of time), all passengers disembark and queue to ride again. The coaster has only one track, so wagons cannot overtake each other. They always arrive in the same order they departed in.

Implement passengers and wagons as nodes that require sychronization.

- When a wagon arrives, c passengers should call a procedure `i_am_boarding`()
- A full wagon should call a procedure `departure`()
- After the end of the ride, the same c passengers should call a procedure `i_am_disembarking`()

Comment

Start with $n = 1$, and extend the solution from there to multiple wagons. The solution should avoid starvation of passengers (every waiting passenger is getting a ride at some point) and should be fair (every passager is riding as often as possible). Highlight these aspects in your presentation.

You can run more than one passenger node / wagon node per physical machine, but passengers and wagons should not be on the same machines.

Group 3
Sleeping Barber problem

In a barber shop there is a barber, a single barber's chair in which clients are serviced and *3* waiting room chairs.

- The barber sleeps, if there are no clients in the shop.
- If a client arrives while the barber is sleeping, the client wakes the barber, takes seat in the barber's chair and the barber starts working.
- If a client arrives while the barber is working and there is an empty waiting room chair, the client will take a seat in the waiting room.
- If a client arrives while the barber is working and all waiting room chairs are occupied, the client will leave immediately.
- Once the barber serviced a client, the client will leave and either a waiting client takes seat in the barber's chair, or the barber goes to sleep if no one is waiting.

Implement barber and clients as nodes that require sychronization. Whether the chairs are just resources or active nodes is up to you.

No client should be starving in a waiting room chair (every client get served at some point) and the choice among multiple clients in the waiting room should be fair.

Behavior should be demonstrated with at least 5 customers in need of a haircut.

Group 4
The coffee machine

On a corridor there is a coffee machine, customers who buy coffee at the machine using an app, and a supplier that refills the coffee machine.

- The machine can handle only one task at a time: giving out one coffee or being refilled by the supplier. The end of each task is announced by the machine to the client or supplier.
- After accepting a coffee buy order, the client must provide payment for the coffee on request of the machine.
- During the execution of a task the machine places new coffee orders in a queue. After finishing its current task, the machine will execute the next request in a first-come-first-serve order.
- If the machine is empty, it informs the supplier about the need for a refill and will not serve anymore coffee to clients.
- The supplier can refill a machine that is not empty.

Implement coffee machine, suppliers and clients as nodes of a distributed system that require sychronization.

The demonstration should include several customers (3+) and not enough coffee for all of them, so that there is a need for a refill.

Group 5
Matchmaking

Imagine a turn-based online strategy game where two players play matches against each other. A matchmaking server arranges which players are playing against each other based on their skill rating (an arbitrary number).

- If a player wants to start a match, the client informs the matchmaking server about that
- If there is no other player of the same skill rating waiting, the server will inform the client that it must wait, and the client will be placed in a queue.
- If there is another player of the same skill level waiting, server will match both players and send them a message containing the address of the server that will host the actual game (not part of this problem).
- If a player is waiting to be matched, the player can cancel the matchmaking, which should remove them from the waiting list.
- If a player is waiting for longer than time *t* to get matched, the server will match that player with another player of a different skill level and send them a message containing the address of the server that will host the actual game (not part of this problem).
- A player playing a match will take a random timeout/break after the match before queuing up again.

Implement server and clients as nodes of a distributed system that require sychronization.

The demonstration should include several players (4+) of two different skill ratings wanting to play matches.

Group 6
Air traffic control

The scenario is about an air traffic controller (ATC) at a single runway, regional airport, which handles all arriving and departing air traffic.

- Any arriving or departing aircraft will contact ATC with the request for runway access.
- If the runway is currently not cleared for any other aircraft, ATC will grant access.
- If the runway is occupied, ATC will instruct the aircraft to enter a holding pattern (arrivals) or wait on the taxi way (departures) until ATC informs the next aircraft to either land or take-off.
- Any aircraft that got ATC clearance to either land or depart from the runway will do so within time t and will inform ATC once they departed/vacated the runway.
- Any arriving aircraft in the holding pattern will declare emergency after some (longer) time interval due to dwindling fuel reserves and may declare emergency for other reasons.
- If an arriving aircraft declares emergency, it will receive the next slot to access the single runway.

Implement the ATC and aircrafts as nodes of a distributed system that require sychronization.

The demonstration should include several aircrafts (3+) and at least one emergency.