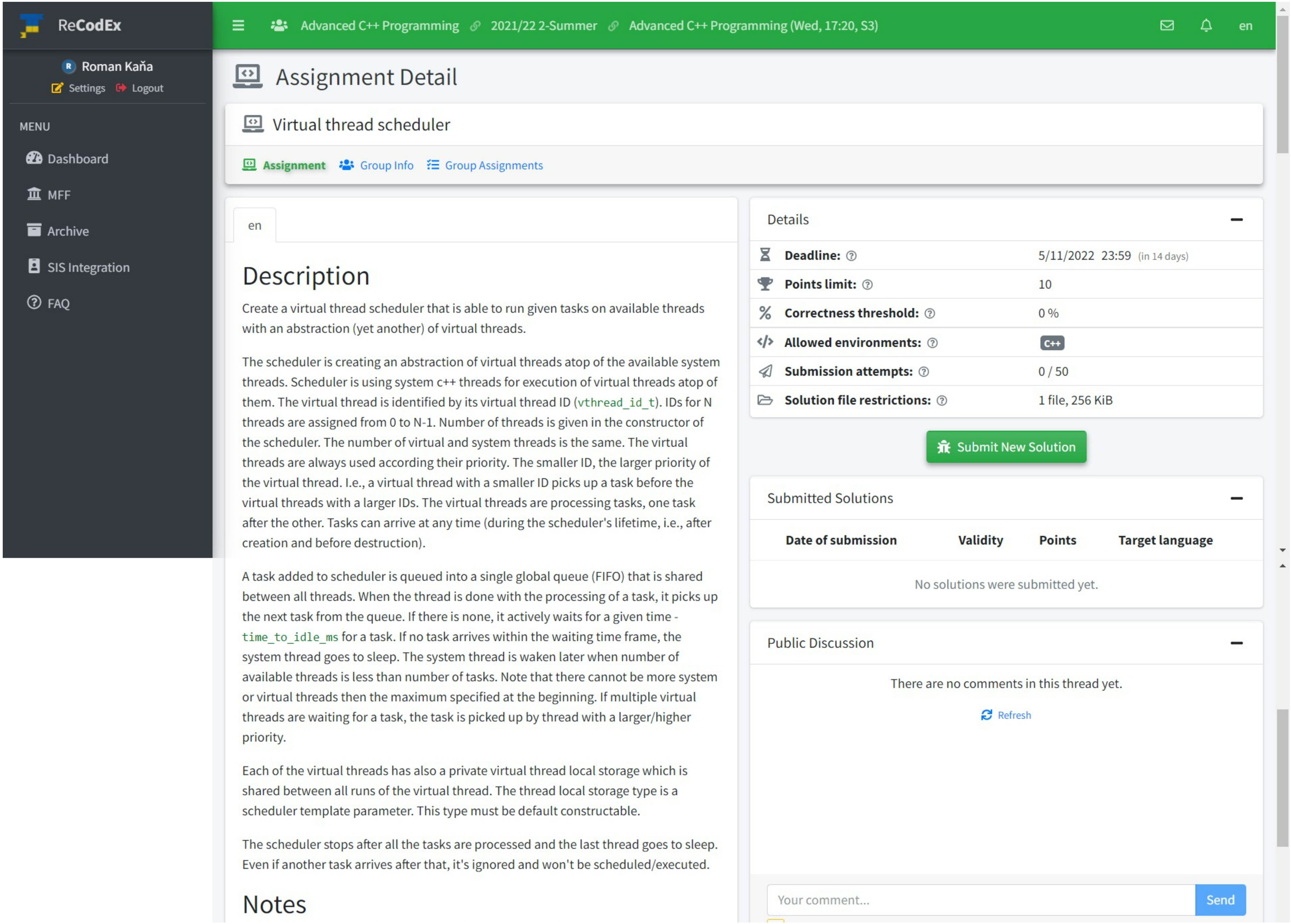


Assignment Detail

Virtual thread scheduler

Description

Create a virtual thread scheduler that is able to run given tasks on available threads with an abstraction (yet another) of virtual threads.

The scheduler is creating an abstraction of virtual threads atop of the available system threads. Scheduler is using system c++ threads for execution of virtual threads atop of them. The virtual thread is identified by its virtual thread ID (`vthread_id_t`). IDs for N threads are assigned from 0 to N-1. Number of threads is given in the constructor of the scheduler. The number of virtual and system threads is the same. The virtual threads are always used according their priority. The smaller ID, the larger priority of the virtual thread. I.e., a virtual thread with a smaller ID picks up a task before the virtual threads with a larger IDs. The virtual threads are processing tasks, one task after the other. Tasks can arrive at any time (during the scheduler's lifetime, i.e., after creation and before destruction).

A task added to scheduler is queued into a single global queue (FIFO) that is shared between all threads. When the thread is done with the processing of a task, it picks up the next task from the queue. If there is none, it actively waits for a given time - `time_to_idle_ms` for a task. If no task arrives within the waiting time frame, the system thread goes to sleep. The system thread is waken later when number of available threads is less than number of tasks. Note that there cannot be more system or virtual threads then the maximum specified at the beginning. If multiple virtual threads are waiting for a task, the task is picked up by thread with a larger/higher priority.

Each of the virtual threads has also a private virtual thread local storage which is shared between all runs of the virtual thread. The thread local storage type is a scheduler template parameter. This type must be default constructable.

The scheduler stops after all the tasks are processed and the last thread goes to sleep. Even if another task arrives after that, it's ignored and won't be scheduled/executed.

Notes

Details

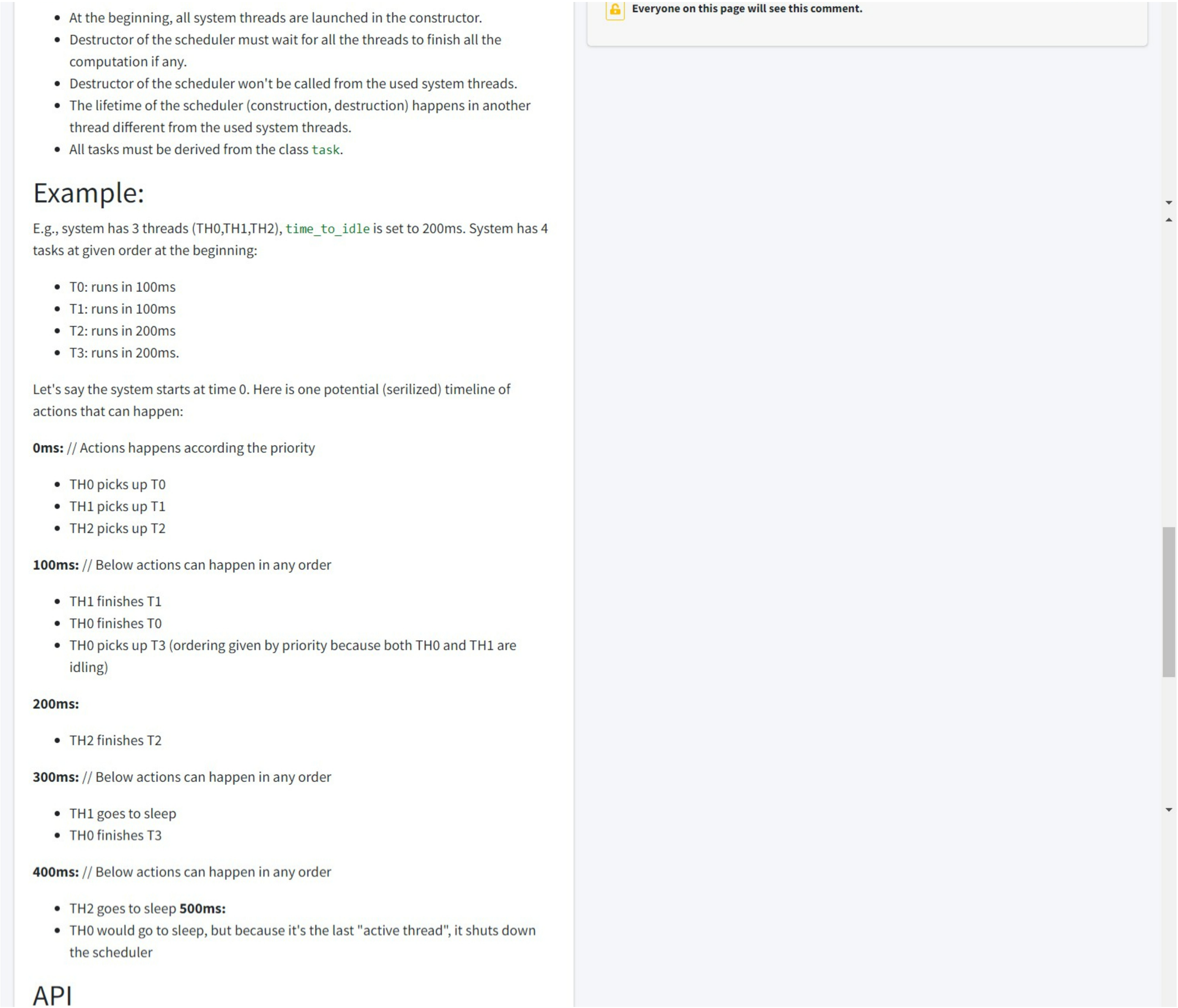
Deadline: ?	5/11/2022 23:59 (in 14 days)
Points limit: ?	10
Correctness threshold: ?	0 %
Allowed environments: ?	C++
Submission attempts: ?	0 / 50
Solution file restrictions: ?	1 file, 256 KiB

Submitted Solutions

Date of submission	Validity	Points	Target language
No solutions were submitted yet.			

Public Discussion

There are no comments in this thread yet.



- At the beginning, all system threads are launched in the constructor.
- Destructor of the scheduler must wait for all the threads to finish all the computation if any.
- Destructor of the scheduler won't be called from the used system threads.
- The lifetime of the scheduler (construction, destruction) happens in another thread different from the used system threads.
- All tasks must be derived from the class `task`.

Example:

E.g., system has 3 threads (TH0,TH1,TH2), `time_to_idle` is set to 200ms. System has 4 tasks at given order at the beginning:

- T0: runs in 100ms
- T1: runs in 100ms
- T2: runs in 200ms
- T3: runs in 200ms.

Let's say the system starts at time 0. Here is one potential (serilized) timeline of actions that can happen:

0ms: // Actions happens according the priority

- TH0 picks up T0
- TH1 picks up T1
- TH2 picks up T2

100ms: // Below actions can happen in any order

- TH1 finishes T1
- TH0 finishes T0
- TH0 picks up T3 (ordering given by priority because both TH0 and TH1 are idling)

200ms:

- TH2 finishes T2

300ms: // Below actions can happen in any order

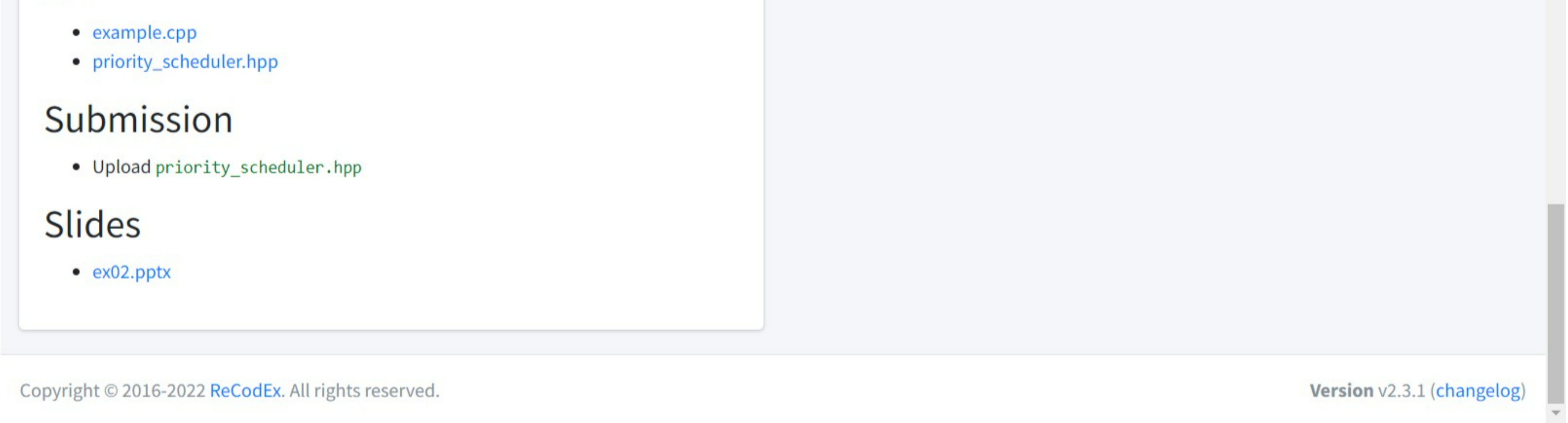
- TH1 goes to sleep
- TH0 finishes T3

400ms: // Below actions can happen in any order

- TH2 goes to sleep **500ms:**
- TH0 would go to sleep, but because it's the last "active thread", it shuts down the scheduler

API

Everyone on this page will see this comment.



- example.cpp
- priority_scheduler.hpp

Submission

- Upload `priority_scheduler.hpp`

Slides

- ex02.pptx