# PCPP Assignment 6

Group name: Thread Heresy
Real names : Nedas Surkus, Niclas Abelsen,
Github username: nesu, niab

December 2022

## Contents

## 12.1

### 1

For this exercises We have added the following states to the server

1. `idleWorkers`
2. `busyWorkers`
3. `tasksToDo`

They are all implemented as queues because every actor and every tasks will be processed eventually. These states are sufficient as they fulfill the requirements for this exercise. The purpose of `idleWorkers` is to keep track of the current worker actors who are ready to receive new tasks from the `taskTodo` queue. The `busyWorkers` are the workers who are currently computing a `Task`. Finally, the `taskTodo` is a map between the `Task` and `Client` that issued the task.

### 2

The constructor initialises a linked list for busy workers, idle workers and `tasksToDo` as well as assigns a number of minimum workers and maximum workers. Afterwards we initialise the minimum amount of workers and add them to the idle worker queue.

This is sufficient as no task can be executed before we execute the create method and this allows us to set up the actor without fear of race conditions.

When implementing elastic workload we can reference min workers and max workers to see if the current worker count inside busy workers and idle workers are within thresholds. Meaning, if a task is finished, we can check if there are too many newly spawned workers by adding idle and busy worker counts and checking if its above minimum. If it is above minimum, we stop the worker and reduce the server resource load.

### 3

When the actor gets a Compute task message , all the tasks inside the message need to be added to the `taskToDo` queue. They are added as a pair of tuples. First value will contain the task itself and the second value contains the client reference that issued this task. This is useful when we need to process tasks if there are not enough workers to process them all at once.

Afterwards, we do a while loop by checking if there are tasks that still are not processed.

If there are tasks and idle workers, we remove the worker from idle workers and add him to the busy workers. Then we take the `tasksToDo` queue and pop the first element and tell the busy worker to execute the task.

Since the `taskToDo` is a queue, we know that it executes based on the FIFO principle, meaning that when we add a task, it will be added to the end of the queue and if we pop a task, it will be taken from the start of the queue so no two requests have priority over the currently executing task.

As well as both idle worker and busy worker are queues, we know that each worker will do an equal amount of work load as workers that have finished the tasks are added in the back of the idle queue and poped from the start of the queue.

If there are no idle workers, we check if busy worker count is not maximum and we generate a new idle worker.Doing this allows us to scale up workload based on the task amount, but not assign infinite amount of workers.

If have no more tasks to assign or we have no idle workers to assign and we assigned the max amount of workers, we break out of the while loop. This allows the actor to begin processing other messages received from clients.

## 4

When an child actor fails, the child sends a childFail signal. When the parent actor receives said signal it executes the method `onChildFailed` that takes a `ChildFailed` message as input.
Firstly, we get the crashed worker, remove it from the busy worker queue and spawn a new worker with the same id.

If the `taskTodo` queue is not empty, we pop the task from the queue and tell the newly spawned worker to execute the task, additionally we re-add the new worker to the busy worker queue. Otherwise, we check if the current amount of workers is below the minimum amount of workers. If that is the case, we add the worker to the idle queue. If none of this is true, we return the behaviour which eliminates the worker if it was not assigned to any queue.

## 5

The `onWorkDone` method firstly check if the pending task queue, `taskToDo` is not empty, if that is the case, pop the task from the queue and assign it to the worker that had just finished it tasks. If there are no pending tasks, we check if the total amount of workers are larger than the minimum amount of workers. If that is the case, we remove the worker from the `busyWorkers` queue and stop the worker. Otherwise, we remove the worker from the queue and add it to `idleWorkers` queue.