Christian-Albrechts-Universität zu Kiel
Institut für Informatik

Priv.-Doz. Dr. Frank Huch, Marius Rasch

Institut für Informatik

3. Übung zur Vorlesung „Concurrent and Distributed Programming“

Abgabe am Monday, 29. April 2019 - 18:00

Aufgabe 1 - Semaphores in Erlang

1 Punkt

Implement the concept of a semaphore in a Erlang module `semaphore`. A semaphore can be created by a function `newSemaphore/1`, where the parameter represents the maximal number of processes in a critical section. After creation the semaphore should be used as usual by functions `p/1` and `v/1`. Additionally the function `l/1` should return the number of threads still allowed to acquire the semaphore.

Aufgabe 2 - Counter in Erlang

1 Punkt

Implement the graphical counter already implemented in Python using Erlang. The start function should expect a list of numbers, which stands for the speed of the counters.

To do this exercise we provide a simple Counter-GUI, so you don't have to implement the GUI itself. By using the function

[]erlang counterGui:start(V,Pid)

you can start a new GUI process, which shows the value `V` and provides the following buttons: Start, Stop, Copy, Clone, Close. The return value of this function is the pid of the process controlling the GUI. If one of the buttons gets pressed, one of the following messages gets send to the processes `Pid`.

[]erlang start, stop, copy, clone, close,GuiPid

When pressing the close button the Pid of the corresponding GUI process is send as well. Additionally the GUI process understands the message `{setValue,V}` to change the current value of the counter.

Hint: To use the provided GUI, Erlang WX is required. This package can be install using the usual package sources (Linux) or using Cygwin (Windows).

Aufgabe 3 - Sieve of Eratosthenes

1 Punkt

In this exercise you should implement the sieve of Eratosthenes using processes in Erlang.

We model the sieve as dynamic growing mesh of Erlang processes. Every sieve process requests a potential prime number by its predecessor. It then filters the multiples of “its” prime number and sends the next possible prime numbers to its successor. To stop the processes generating an infinite number of possible prime numbers, the implementation should be demand-driven. A sieve only sends a next number, if it is asked for.

Your Erlang program should start with a process sending (on request) all integers starting with 2. In Addition you should define a process which requests the next prime number and coordinates the creation of new sieves.

Aufgabe 4 - Chat without server 1 Punkt

In the lecture you saw a chat implemented in Erlang. The architecture of this chat uses a client-server model.

1. Implement an alternative version of the chat, which doesn't need a fixed server. A client can join a chat room if it knows any client which already joined the room. Hence, a chat room is defined by all clients connected to it. Compare the number of messages send using this implementation with the number when using the server-client implementation.
2. The naive implementation of this system has an issue, resulting in inconsistent data between single chat clients. Outline this issue using a small example. Think about a way to solve this issue (without implementing it).