**Christian-Albrechts-Universität zu Kiel**
Institut für Informatik

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

Institut für Informatik

# 4. Übung zur Vorlesung „Concurrent and Distributed Programming“

Abgabe am Monday, 06. May 2019 - 18:00

---

## Aufgabe 1 - Distributed Game in Erlang

1 Punkt

In this exercise you should implement a simple distributed game in Erlang. There are several players playing against each other. In the game window a button appears on a random position. The player who presses the button first wins. To do this exercise we provide a Game-GUI, which can be started by calling

```
1 gameGui:start(ClientPid,X,Y)
```

The GUI sends messages to the process with the pid `ClientPid`. `X` and `Y` specify the size of the window. The GUI understands the following messages:

- `{text_update,Text}`: Shows `Text` in the window.
- `{button_create,ID,X,Y}`: Creates a Button with ID `ID` at position `X`,`Y`. The size of the button is 100x50. If a button already exist, the old button is replaced by the new one. There can only be one button at the time.
- `button_remove`: Removes the button.

The GUI sends the following messages to `ClientPid`:

- `{clicked,ID}`: The button with ID `ID` was clicked.
- `close`: The GUI was closed.

Develop a `game_server` and a `game_client`. Several clients should be able to join the server. All players see the button at the same time at the same place. The player who presses the button first wins this round. The game can run several rounds and you can use the provided text field to show the points for instance. You can decide how the game works in detail. Use the techniques of distributed programing seen in the lecture and make sure to build a robust program.

## Aufgabe 2 - Chat without server

1 Punkt

There is an issue in the sample solutions of the chat without server from last week. If there are two new clients joining the chat at different nodes at the same time the information about all existing clients might be inconsistent. Check if this bug exists in your solution.

In this exercise you shall correct your solution (or the sample solution). If the issues doesn't exist in your solution, explain why it's correct.

A possible solution is to append a list of all clients to every message. This brings a huge overhead. An optimization might be to hash this list (erlang:phash/2) and only request the full list if required.

Make sure your solution is robust.

### Aufgabe 3 - TCP chat in Erlang 1 Punkt

Reimplement the Erlang chat from the lecture (server client chat) without using the techniques for distributed programming provided by Erlang. For communication between client and server use native TCP instead. Of course the processes of on participant are allowed to communicate using messages as normal.

### Aufgabe 4 - Extension of RemoteChan 1 Punkt

In the lecture we implemented a RemoteChan using TCP. Your even discussed a few specialities you're going to implement in this exercise.

1. For a Chan only used local, no portListener should be created .
2. A remoteChan returning to its local node should be converted into a localChan again.