Fehlertolerante Programme mit Erlang

Erlang–Prozesse können sich gegenseitig überwachen. Dazu werden mit `link` Verbindungen zwischen Prozessen geknüpft.

17. **Prozesse beenden**
    Bitte erweitern Sie den `counter`–Prozess um folgende Funktionalität:

    a) die Behandlung einer `stop`–Nachricht, bei dem er regulär (mit Grund `normal`) endet.

    b) das Beenden mit dem Grund `underflow`, falls der Counter (durch `down`–Nachrichten) kleiner als null wird.

    c) das Protokollieren unbekannter Nachrichten, die an den `counter`–Prozess gesendet werden.

18. **Überwachung von Prozessen**

    a) Schreiben Sie bitte einen normalen Prozess `echo`, der die empfangenen Nachrichten einfach ausgibt und der mit einem `counter`–Prozess verbunden ist. Was geschieht, wenn der `counter`–Prozess regulär oder irregulär beendet wird.

    b) Verwandeln Sie den `echo`–Prozess in einen System–Prozess. Wie ändert sich das Verhalten?

19. **Programmierte Überwachung mit `on_exit`**

    a) Verwenden Sie bitte die in der Vorlesung vorgestellte Funktion `on_exit`, um den erweiterten `counter`–Prozess zu überwachen und eine Meldung auszugeben, sobald er beendet wird.

    b) Beenden Sie den überwachten `counter`–Prozess regulär (mit normal) und irregulär (mit `underflow`). Werden beide Fälle durch die Überwachung erkannt?

    c) Erweitern Sie die Überwachung bitte so, dass normales Beenden und Beenden mit `underflow` unterschiedlich behandelt werden.

20. **Automatisches Neustarten mit `keep_alive`**

    a) Sorgen Sie bitte mit Hilfe der in der Vorlesung vorgestellten Funktion `keep_alive` dafür, dass der `counter`–Prozess sich unter dem Namen `ctr` registiert und neu gestartet wird, falls er sich beendet.

    b) Überwachen Sie bitte eine Instanz des `counter`–Prozesses wie in Aufgabe 19a und beobachten Sie wann der `counter`–Prozess beendet und dann neugestartet wird.

    c) Beobachten Sie mit `whereis(ctr)` wie sich beim Neustarten des `counter`–Prozesses der an den Namen `ctr` gebundene Prozess ändert.

    d) Um die in `keep_alive` verborgene Race–Condition zu beobachten, verlängern wir in `keep_alive` die Zeit zwischen `register` und `on_exit` und geben Hilfsinformationen aus:

```
keep_alive(Name, Fun) ->
   register(Name, Pid = spawn(Fun)),

   receive after 10000 -> ok end,
   io:format("Pid␣~p␣is_process_alive:␣~p~n",
       [Pid, erlang:is_process_alive(Pid)]),

   on_exit(Pid, fun(_Why) -> keep_alive(Name, Fun) end).
```

    Senden Sie bitte in schneller Folge `stop`–Nachrichten an den unter `ctr` registrierten Prozess und beobachten Sie bitte das Verhalten von `keep_alive`.

    Wie könnte eine korrigierte Version von `keep_alive` aussehen?