Verteilte Systeme 2012: 7. Übungszettel

Max Michels
Schintke, Schütt — Philipp Borgers
21.06.2012 — Sascha Schönfeld

1 Maekawa-Algorithmus in Erlang

1.a Code

```
1  -module(maekawa).
2  -export([init/0, initProc/0]).
3
4  %% feel free to adjust the init function
5
6  init() ->
7      %%6 processes and 3 groups
8      P1 = spawn(maekawa, initProc, []),
9      P2 = spawn(maekawa, initProc, []),
10     P3 = spawn(maekawa, initProc, []),
11     P4 = spawn(maekawa, initProc, []),
12     P5 = spawn(maekawa, initProc, []),
13     P6 = spawn(maekawa, initProc, []),
14     G1 = [P1,P2,P3],
15     G2 = [P2,P4,P5],
16     G3 = [P5,P6,P1],
17     P1 ! {setGroup, G1},
18     P2 ! {setGroup, G2},
19     P3 ! {setGroup, G1},
20     P4 ! {setGroup, G2},
21     P5 ! {setGroup, G3},
22     P6 ! {setGroup, G3},
23     P1 ! enterCriticalSection,
24     P3 ! enterCriticalSection,
25     P6 ! enterCriticalSection,
26     'initialization done'.
27
28 %%tell a process which group it belongs to
29 initProc() ->
30     InitState = {_State = released, _Voted = false, _Requests = [], _Replies = 0},
31     receive
32         {setGroup, V} ->
33             io:format("initializing ~p with ~p~n",[self(), V]),
34             loop(InitState, V);
35         _ ->
36             io:format("init for ~p failed~n",[self()])
37     end.
38
39 %%main loop
40 loop(State, V) ->
41     {ProcState, Voted, Requests, Replies} = State,
42     NewState =
43         case ProcState of
44             held ->
45                 %%we can now safely enter the critical sections
46                 criticalSection(),
47                 [GroupMember ! release || GroupMember <- V],
48                 {_ProcState = released, Voted, Requests, Replies};
49             wanted ->
50                 if
51                     Replies =:= length(V) ->
52                         {_ProcState = held, Voted, Requests, 0};
53                     true ->
54                         processMessages(State, V)
```

Verteilte Systeme 2012: 7. Übungszettel

Max Michels
Schintke, Schütt — Philipp Borgers
21.06.2012 — Sascha Schönfeld

```
55                  end;
56              released ->
57                  processMessages(State, V)
58          end,
59      %io:format("~p: state is ~p~n",[self(), NewState]),
60      loop(NewState, V).
61
62  %%message processing
63  processMessages(State, V) ->
64      {ProcState, Voted, Requests, Replies} = State,
65          receive
66              enterCriticalSection ->
67                  io:format("~p: trying to enter critical section~n",[self()]),
68                  [GroupMember ! {request, self()} || GroupMember <- V],
69                  io:format("~p: sent out requests to group ~n", [self()]),
70                  {_State = wanted, _Voted = false, Requests, Replies};
71              {request, Pid} ->
72                  case State of
73                      {_ProcState = held, _Voted, _Requests, _Replies} ->
74                          io:format("~p: queuing request from ~p~n",[self(), Pid]),
75                          {ProcState, Voted, Requests ++ [Pid], Replies};
76                      {_ProcState, _Voted = true, _Requests, _Replies} ->
77                          io:format("~p: queuing request from ~p~n",[self(), Pid]),
78                          {ProcState, Voted, Requests ++ [Pid], Replies};
79                      _ ->
80                          io:format("~p: Sending reply to ~p~n", [self(), Pid]),
81                          Pid ! reply,
82                          {ProcState, _Voted = true, Requests, Replies}
83                  end;
84              release ->
85                  case Requests of
86                      [] ->
87                          {ProcState, _Voted = false, [], Replies};
88                      [H | T] ->
89                          io:format("~p: Sending reply to ~p~n", [self(), H]),
90                          H ! reply,
91                          {ProcState, _Voted = true, T, Replies}
92                  end;
93              reply ->
94                  io:format("~p: Received reply number ~p~n", [self(), Replies+1]),
95                  {ProcState, Voted, Requests, Replies + 1};
96              _ ->
97                  io:format("~p: WARNING i don't understand this message: ~~n",[self()]),
98                  State
99          end.
100
101 %%the critical section
102 criticalSection() ->
103     io:format("~p: I'm in the critical section. Hope noone's around...~n", [self()]),
104     timer:sleep(1000),
105     io:format("~p: I'm leaving the critical section~n", [self()]).
```

Verteilte Systeme 2012: 7. Übungszettel

Max Michels
Schintke, Schütt — Philipp Borgers
21.06.2012 — Sascha Schönfeld

1.b Erläuterung

Wir haben den Maekawa-Algorithmus für eine flexible Anzahl von Prozessen implementiert. In der Funktion init() können beliebig viele Prozesse gespawnt werden. Damit der wechselseitige Ausschluss zwischen den Prozessen funktioniert, muss jedem Prozess eine Prozessgruppe zugeordnet werden. Dies kann nach der Initialisierung des Prozesses mit eine Nachricht setGroup, GROUP an den jeweiligen Prozess erfolgen. Anschließend können Prozesse mit einer Nachricht der Form enterCriticalSection in den kritischen Bereich geschickt werden. Der kritische Bereich wird von der Funktion criticalSection() simuliert.

Der Maekawa-Algorithmus stellt in der Funktion loop() sicher, dass stets nur ein Prozess im kritischen Bereich landet. Das heißt, dass wenn ein Prozess den Zustand held hat, die anderen Prozesse warten müssen und entsprechende Anfragen gebuffert werden. Für einen Prozess gibt es drei Zustände, in dem er sein kann. Entweder ist ein prozess in der kritischen Sektion (held), oder er wartet auf eine Freigabe (wanted), oder aber er befindet sich im normalen Ablauf (released). Im Zustand held und wanted werden stets Nachrichten abgeareitet, wohingegen im held Zustand nicht. Der eigentliche Algorithmus sieht auch im Held Zustand eine Abarbeitung der Nachrichten vor, jedoch führt unsere Variante auch zum wechselsetitigen Ausschluss.

Verteilte Systeme 2012: 7. Übungszettel

Max Michels
Schintke, Schütt — Philipp Borgers
21.06.2012 — Sascha Schönfeld

1.c Testlauf

```
maekawa:init().
initializing <0.156.0> with [<0.156.0>,<0.157.0>,<0.158.0>]
initializing <0.157.0> with [<0.157.0>,<0.159.0>,<0.160.0>]
initializing <0.158.0> with [<0.156.0>,<0.157.0>,<0.158.0>]
initializing <0.159.0> with [<0.157.0>,<0.159.0>,<0.160.0>]
initializing <0.160.0> with [<0.160.0>,<0.161.0>,<0.156.0>]
initializing <0.161.0> with [<0.160.0>,<0.161.0>,<0.156.0>]
'initialization done'
<0.156.0>: trying to enter critical section
<0.158.0>: trying to enter critical section
<0.158.0>: sent out requests to group
<0.161.0>: trying to enter critical section
<0.156.0>: sent out requests to group
<0.157.0>: Sending reply to <0.158.0>
<0.158.0>: Sending reply to <0.158.0>
<0.158.0>: queuing request from <0.156.0>
<0.161.0>: sent out requests to group
<0.158.0>: Received reply number 1
<0.156.0>: Sending reply to <0.158.0>
<0.157.0>: queuing request from <0.156.0>
<0.158.0>: Received reply number 2
<0.160.0>: Sending reply to <0.161.0>
<0.161.0>: Sending reply to <0.161.0>
<0.156.0>: queuing request from <0.156.0>
<0.158.0>: Received reply number 3
<0.158.0>: I'm in the critical section. Hope noone's around...
<0.156.0>: queuing request from <0.161.0>
<0.161.0>: Received reply number 1
<0.161.0>: Received reply number 2
<0.158.0>: I'm leaving the critical section
<0.158.0>: Sending reply to <0.156.0>
<0.156.0>: Sending reply to <0.156.0>
<0.156.0>: Received reply number 1
<0.156.0>: Received reply number 2
<0.157.0>: Sending reply to <0.156.0>
<0.156.0>: Received reply number 3
<0.156.0>: I'm in the critical section. Hope noone's around...
<0.156.0>: I'm leaving the critical section
<0.156.0>: Sending reply to <0.161.0>
<0.161.0>: Received reply number 3
<0.161.0>: I'm in the critical section. Hope noone's around...
<0.161.0>: I'm leaving the critical section
```