Damson and Fireflies

Aims and Outcomes

● Aim to learn
  – A handy language to readily program concurrent algorithms.
  – Firefly synchronisation

● Outcomes
  – Ability to demonstrate firefly synchronisation.
  – Ability to demonstrate Peterson’s critical region algorithm for distributed systems.
Peterson mutual exclusion 1

2 player version

initially flagi=0 and turn is shared
Player 1                 Player 2
flag1 = 1                flag2 = 1
turn = 1                  turn = 2
waitfor                  waitfor
flag2=0 or                flag1=0 or
turn = 2                  turn = 1
                          critical region
flag1=0                  flag2=0
Peterson mutual exclusion 2

\[N\] player version Shared variables

\[k = \{1, \ldots, N-1\}\]
\[\text{turn}(k) \text{ from } \{1, \ldots, N\}\]
\[i = \{1, \ldots, N\}\]
\[\text{flag}(i) \text{ from } \{0, \ldots, N-1\} \text{ initially 0}\]

Process \(i\)
for \(k=1\) to \(N-1\) do
    \[\text{flag}(i)=k\]
    \[\text{turn}(k)=i\]
    wait for
    \[\text{flag(}\forall j \text{ not } i) < k\]
    or \(\text{turn}(k)\) not \(i\)
critical region
\[\text{flag}(i)=0\]

Program this in Java(using threads) and Damson
Firefly function

\[ f(\phi) = b^{-1} \ln(1 + [e^{b} - 1]\phi) \]

\[ g(x) = \frac{e^{bx} - 1}{e^{b} - 1} \]

\[ x_i(t) = 1 \Rightarrow x_j(t^+) = \min(1, x_j(t) + \epsilon) \]

Graph of the function \( x = f(\phi) \)

*state as a function of phase*
b=3; e = 0.01; n = 10000;
diffP = zeros(1,n); diffX = zeros(1,n);
dphi = 0.001; phiA = 1/2; phiB = 2/3;
for i = 1:n
    phiA = phiA + dphi;
    phiB = phiB + dphi;
    xA = f(b,phiA);
    xB = f(b,phiB);
    if xA >= 1
        xA = 0;
        xB = min(xB + e,1);
        if xB >= 1
            xB = 0;
        end
    end
    if xB >= 1
        xB = 0;
        xA = min(xA + e,1);
        if xA >= 1
            xA = 0;
        end
    end
    phiA = g(b,xA);
    phiB = g(b,xB);
    diffP(i) = phiA-phiB;
    diffX(i) = xA - xB;
end
figure(1);
plot(diffP);
title('phase difference');
figure(2)
plot(diffX);
title('state difference');
p = linspace(0,1,1000);
mx = f(b,p);
figure(3)
plot(p,mx);
title('Graph of the function x = f(\phi)');
xlabel('\phi');
ylabel('x');
function x = f(b,phi)
    % f is the mapping of phase phi to state x
    % in the Mirollo model
    x = log(1+(exp(b)-1)*phi)/b;
    if phi >= 1
        x = 1;
    end
end

function phi = g(b,x)
    % This is the mapping of state x to phase phi
    % for the Mirollo model
    phi = (1-exp(b*x))/(1-exp(b));
    if x >= 1
        phi = 1;
    end
end
Convergence of state

state difference evolving
Convergence of phase

phase difference evolving