Distributed Mutual Exclusion

source: Sukumar Ghosh's Distributed Systems, an algorithmic approach

- Introduction
- Solutions using Message Passing
- Token Passing Algorithms

Why do we need Distributed Mutual Exclusion?

- to deal with resource sharing
- to avoid concurrent updates on shared data
- to control the grain of atomicity
- to gain access to ethernet devices (ex: CSMA/CD protocol to resolve bus contention in ethernets)
- to avoid collisions in wireless broadcasts
- many other reasons...

An example: bank account transaction

```
shared n: integer (n is a shared account)

Process P                        Process Q
Account receives amount nP       Account receives amount nQ
Computation: n = n + nP:         Computation: n = n + nQ:
   P1. Load Reg_P, n                Q1. Load Reg_Q, n
   P2. Add Reg_P, nP                Q2. Add Reg_Q, nQ
   P3. Store Reg_P, n               Q3. Store Reg_Q, n
```

An example: bank account transaction

Possible interleaves of executions of P and Q:

- Giving expected result nP + nQ:
  - P1, P2, P3, Q1, Q2, Q3
  - Q1, Q2, Q3, P1, P2, P3
- Giving incorrect result n=n+nP:
  - Q1, P1, Q2, P2, Q3, P3
  - Q1, Q2, P1, P2, Q3, P3
  - Q1, P1, P2, Q2, Q3, P3
  - P1, Q1, P2, Q2, Q3, P3
  - P1, P2, Q1, Q2, Q3, P3
- Giving incorrect result n=n+nQ:
  - P1, Q1, P2, Q2, P3, Q3
  - P1, P2, Q1, Q2, P3, Q3
  - P1, Q1, Q2, P2, P3, Q3
  - Q1, P1, Q2, P2, P3, Q3
  - Q1, Q2, P1, P2, P3, Q3

Critical Section (CS)

Each process, before entering a CS acquires authorization. If it gains authorization, blocks other processes of executing the same CS, executes it, and releases it.

Mutual Exclusion for 2 processes, for shared memory, Knuth's protocol

Process P_i: (3 shared variables: A[0], A[1], B)
$i \in 0, 1$
other process $j = 1 - i$

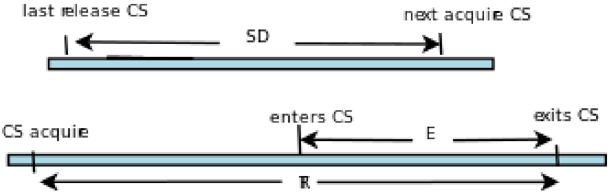
```
loop
   non critical section;                          \
   loop                                            |
     A[i] := 1;                                    |
     await B == i OR A[j] == 0;                    |  entry section
     A[i] := 2;                                    |
     if A[j] != 2 then break;                      |
   end loop;                                       |
   B := i;                                        /
   critical section;
   B := j;                                        \   exit section
   A[i] := 0                                      /
end loop
```

Correctness Conditions

- ME1: Mutual Exclusion
  - at most one process can remain in the CS at any time: safety property!
- ME2: Freedom from deadlock
  - at least one process is eligible to enter the CS: liveness property!
- ME3: Fairness
  - every process will eventually succeed in entering the CS: NO starvation property!
- A measure of fairness: bounded waiting
  - specifies an upper bound on the number of times a process waits for its turn to enter the CS: n-fairness (n is the maximum number of rounds)
  - whe n=0, FIFO fairness

How to Measure Performance

- Number of msgs per CS invocation
- Fairness
- Synchronization delay (SD)
- Response Time (RT)
- System Throughput (ST)
  - $ST = \frac{1}{SD+E}$, where E is the average CS execution time



Fault Tolerance

- not much true for the algorithms presented here

Message Passing Algorithms

Assumptions:

- n processes ($n > 1$), numbered 0, 1, . . ., n-1, P_i communicating by sending/receiving messages
- topology: graph completed connected
- each P_i periodically:
  1. enters CS
  2. execute CS code
  3. exits CS code
- attend ME1, ME2 and ME3

Message Passing Algorithms

Easy Solution???

Message Passing Algorithms

Easy Solution???

- Centralize :-)