Group Mutual Exclusion (GME) Algorithms

-A simple local-spin GME &
a space-efficient FCFS GME

By Carrie Chu *April, 2009*

The problem

- A process requests a “session”.
- Processes requesting the same session can be in CS simultaneously.
- Processes requesting different sessions can not.
- Usual ME algorithm can’t be directly applied to solve the problem.

E.g. a CD jukebox

GME model

A group mutual exclusion process:

repeat
  NCS
  Try section
  CS
  Exit section
forever

- The problem is to design Try and Exit sections, s.t. certain properties can be satisfied.

GME properties

- **(P1)** Mutual exclusion: If two processes are in the CS at the same time, then they request the same session.
- **(P2)** Lockout freedom: If a process enters the Try section, then it eventually enters the CS.
- **(P3)** Bounded exit: If a process enters the Exit Section, then it enters the NSC within a bounded number of its own steps.
- **(P4)** Concurrent entering: If a process *i* requests a session and no process requests a different session, then *i* enters the CS within a bounded number of its own steps.

Two GME algorithms

- Patrick Keane and Mark Moir. A simple local-spin group mutual exclusion algorithm. In *Proceedings of the 18th annual ACM Symposium on Principles of Distributed Computing*, pages 23-32, Atlanta, Georgia, United States, 1999. ACM.
- Srdjan Petrovic. Space-efficient FCFS group mutual exclusion. *Information Processing Letters*, 95(2): 343-350, July 2005.

Algorithm 1: local-spin GME

- Satisfy P1-P3, and a weak P4 (concurrent occupancy)
- It uses:
  - An exclusive lock *M* (implemented by any ME)
  - A process waiting queue *Q*
- Each process gets a spin location in an boolean array of N processes *wait*, in which the process can wait to enter CS.

shared variables

M: **lock**; *Session*, *Num*: **integer**; *Q*: **queue of** 0..*N*-1;

Wait: **array** [0..*N*-1] **of boolean**; *Need*: **array** [0..*N*-1] **of integer**

local variables

t, *v*: **integer**;

initially

$Num = 0 \ \wedge \ Session = 1 \ \wedge \ Q = \emptyset$

0: t=

```
1:   Wait[p] := false;
2:   Need[p] := t;
3:   Acquire(M);
4:   if Session = t ∧ Q = ∅ then
5:       Num := Num+1
6:   else if Session ≠ t ∧ Num = 0 then
7:       Session := t;
8:       Num := 1
     else
9:       Wait[p] := true;
10:      Enqueue(Q, p)
     fi;
11:  Release(M);
12:  while Wait[p] do od;
```

Try section

<<Attend session *t*>>

```
13:  Acquire(M);
14:  Num := Num-1;
15:  if Q ≠ ∅ ∧ Num = 0 then
16:      Session:= Need[Head(Q)];
17:      for each v ∈ Q do
18:          if Need[v] = Session then
19:              Delete(Q, v);
20:              Num := Num+1;
21:              Wait[v] := false
     fi od fi;
22:  Release(M)
```

23: go to 0

Exit section

A simple local-spin group mutual exclusion algorithm. Code is shown for process *p*.

An example for algorithm 1

Process	i	j	k	l	m	n
Session	s1	s1	s2	s2	s1	s2

Result: i, j $\rightarrow$ k, l, n $\rightarrow$ m

Algorithm 2: space-efficient FCFS GME

- Satisfy P1-P4, and FCFS (first come first served)

  (P5) FCFS: If a process *i* completes the doorway before process *j* enters the doorway and the two processes request different sessions, then *j* doesn't enter the CS before *i*.

- Space efficient: $\Theta(N)$ without deadlock

Algorithm 2: space-efficient FCFS GME

- Transformed from Lycklama-Hadzilacos ME algorithm [doi:10.1145/115372.115370]
- Not use lock
- Shared variables are all arrays of N processes. Each cell owned by a process, has a single writer (its owner) and multiple readers.
- Modular composition of two parts: FCFS+ME

Shared variables for each $i \in \{1, 2, \ldots, N\}$

- $\mathtt{session}_i$: integer
- $\mathtt{turn}_i$: $\{0, 1, \ldots, 11\}$
- $\mathtt{competing}_i$: boolean

Local variables

- *turn_snap*: array $[1 \ldots N]$ of $\{0, 1, \ldots, 11\}$

```
repeat
    Remainder Section
    session_i = mysession
    for j = 1 to N do turn_snap[j] = turn_j
    if conflict(mysession)
        turn_i = (turn_i + 1) mod 12
    for j = 1 to N do
        wait until (session_j ∈ {0, mysession})
                   ∨ (turn_snap[j] ≠ turn_j)
L:  competing_i = true
    for j = 1 to i − 1 do
        if competing_j ∧ (session_j ∉ {0, mysession})
            competing_i = false
            wait until (¬competing_j)
                       ∨ (session_j ∈ {0, mysession})
            go to L
    for j = i + 1 to N do
        wait until (¬competing_j)
                   ∨ (session_j ∈ {0, mysession})
    CS
    competing_i = false
    session_i = 0
forever
```

Lines from "session_i = mysession" to the second "wait until (session_j ∈ {0, mysession}) ∨ (turn_snap[j] ≠ turn_j)" → **FCFS**

Lines from "L: competing_i = true" to the final "wait until" of the try section → **ME**

Whole block (FCFS + ME) → **Try section**

$\mathtt{competing}_i = \mathit{false}$ → **ME**

$\mathtt{session}_i = 0$ → **FCFS**

Both lines → **Exit section**

Space efficient FCFS algorithm – code for process i

FCFS in Try section

```
session_i = mysession
for j = 1 to N do turn_snap[j] = turn_j
if conflict(mysession)
    turn_i = (turn_i + 1) mod 12
for j = 1 to N do
    wait until (session_j ∈ {0, mysession})
                  ∨ (turn_snap[j] ≠ turn_j)
```

→ **doorway**

Process	i	j	k	l	m	n
Session	s1	s1	s2	s2	s1	s2

Result: i, j → k, l → m → n

ME in Try section

```
L: competing_i = true
   for j = 1 to i − 1 do
      if competing_j ∧ (session_j ∉ {0, mysession})
         competing_i = false
         wait until (¬competing_j)
                     ∨ (session_j ∈ {0, mysession})
         go to L
   for j = i + 1 to N do
      wait until (¬competing_j)
                  ∨ (session_j ∈ {0, mysession})
```

Process	i	j	k
Session	s1	s2	s1

Result: i, k $\rightarrow$ j

Characteristics comparison

	Local Spin	**Space-efficient FCFS**
Use Lock	Yes	No
Access Order	Capturing	FCFS
GME Properties		
Mutual Exclusion	√	√
Lockout Freedom	√	√
Bounded Enter	√	√
Concurrent Entering	Weak (concurrent occupancy)	√
Complexity	O(N)	O(N)
Remote references	bounded	NUMA: unbounded CC: O(N)