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:

\begin{verbatim}
repeat
  NCS
  Try section
  CS
  Exit section
forever
\end{verbatim}

- 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


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 \textit{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
  $Num = 0 \land Session = 1 \land Q = \emptyset$

0: $t =$

1: $Wait[p] := false$;
2: $Need[p] := t$;
3: $Acquire(M)$;
4: if $Session = t \land Q = \emptyset$ then
5:   $Num := Num + 1$
6: else if $Session \neq t \land Num = 0$ 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;

<<Attend session $t$>>

13: $Acquire(M)$;
14: $Num := Num - 1$;
15: if $Q \neq \emptyset \land Num = 0$ then
16:   $Session := Need[Head(Q)]$
17: for each $v \in Q$ do
18:   if $Need[v] = Session$ then
19:     $Delete(Q, v)$
20:     $Num := Num + 1$
21:     $Wait[v] := false$
22: fi od fi;
23: $Release(M)$

Try section

Exit section

A simple local-spin group mutual exclusion algorithm. Code is shown for process $p$. 
An example for algorithm 1

<table>
<thead>
<tr>
<th>Process</th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l</th>
<th>m</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td>s1</td>
<td>s1</td>
<td>s2</td>
<td>s2</td>
<td>s1</td>
<td>s2</td>
</tr>
</tbody>
</table>

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

- 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 ∈ {1, 2,..., N}
   session_i: integer
   turn_i: {0, 1,..., 11}
   competing_i: boolean

Local variables
   turn_snap: array [1...N] of {0, 1,..., 11}

repeat
1:   Remainder Section

2:   session_i = mysession
3:   for j = 1 to N do turn_snap[j] = turn_j
4:   if conflict(mysession)
5:       turn_i = (turn_i + 1) mod 12
6:   for j = 1 to N do
7:       wait until (session_j ∈ {0, mysession})
     ∨ (turn_snap[j] ≠ turn_j)

8L: competing_i = true
9:   for j = 1 to i - 1 do
10:  if competing_j ∧ (session_j ≠ {0, mysession})
11:    competing_i = false
12:   wait until (¬ competing_i)
     ∨ (session_j ∈ {0, mysession})
13:   go to L
14: for j = i + 1 to N do
15:   wait until (¬ competing_i)
     ∨ (session_j ∈ {0, mysession})

16: CS

17: competing_i = false
18: session_i = 0

forever

Space efficient FCFS algorithm – code for process i
FCFS in Try section

\begin{verbatim}
2: session_i = mysession
3: for j = 1 to N do turn_snap[j] = turn_j
4: if conflict(mysession)
5:   turn_i = (turn_i + 1) mod 12
6: for j = 1 to N do
7:   wait until (session_j \in \{0, mysession\})
     \land (turn_snap[j] \neq turn_j)
\end{verbatim}

<table>
<thead>
<tr>
<th>Process</th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l</th>
<th>m</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td>s1</td>
<td>s1</td>
<td>s2</td>
<td>s2</td>
<td>s1</td>
<td>s2</td>
</tr>
</tbody>
</table>

Result: i, j → k, l → m → n
ME in Try section

```
8L: competing_i = true
9:  for j = 1 to i - 1 do
10:     if competing_j ∧ (session_j ∉ {0, mysession})
11:        competing_j = false
12:        wait until (¬competing_j)
13:           ∨ (session_j ∈ {0, mysession})
14:          go to L
15:     for j = i + 1 to N do
16:        wait until (¬competing_j)
17:           ∨ (session_j ∈ {0, mysession})
```

<table>
<thead>
<tr>
<th>Process</th>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td>s1</td>
<td>s2</td>
<td>s1</td>
</tr>
</tbody>
</table>

Result: i, k → j
## Characteristics comparison

<table>
<thead>
<tr>
<th></th>
<th>Local Spin</th>
<th>Space-efficient FCFS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use Lock</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Access Order</strong></td>
<td>Capturing</td>
<td>FCFS</td>
</tr>
<tr>
<td><strong>GME Properties</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mutual Exclusion</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Lockout Freedom</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Bounded Enter</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Concurrent Entering</td>
<td>Weak</td>
<td>√</td>
</tr>
<tr>
<td>(concurrent occupancy)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Complexity</strong></td>
<td>O(N)</td>
<td>O(N)</td>
</tr>
<tr>
<td><strong>Remote references</strong></td>
<td>bounded</td>
<td>NUMA: unbounded</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CC: O(N)</td>
</tr>
</tbody>
</table>