Group Mutual Exclusion (GME) Algorithms

-- Verification of the local-spin GME and the space-efficient FCFS GME

By Carrie Chu *May, 2009*

Problem review

- A process requests a "session".
- Processes requesting the same session can be in CS simultaneously.
- Processes requesting different sessions can not.
- A group mutual exclusion process:

  repeat

  NCS: sleep(5)

  Try section

  CS: sleep(5)

  Exit section

  forever

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 review (1)

- Local-spin algorithm:
  - Use an exclusive lock to protect access to other shared variables
  - The lock is implemented by a semaphore.

```
Semaphore s_lock = new Semaphore(1);

s_lock.acquire();
// access shared variables
s_lock.release();
```

Algorithm review (2)

A process in local-spin algorithm does:

Decide a session

Try Section

Acquire lock s_lock
Check if can go to CS
Release lock s_lock
Busy wait if can not enter CS

CS

Exit Section

Acquire lock s_lock
If it's the last process left the CS and there are still processes waiting to enter CS, establish a new session
Release lock s_lock

Algorithm review (3)

- Space-efficient FCFS algorithm
  - It doesn’t use lock, semaphore, compare-and- swap, compare-and-set atomic mechanisms
    - the code is sequential with some busy waits.
  - Shared variables are owned by each process, each of which has a single writer (its owner) and multiple readers.
    - Shared variables are implemented as private attributes in a process object, with only public read methods.
  - It satisfies property FCFS.
  - Modular composition of two parts: FCFS+ME

Verifications

- Local-spin algorithm
  - group ME property
  - use of lock is essential to ensure group ME
- Space-efficient FCFS algorithm
  - group ME property
  - some codes are essential to avoid deadlock
  - data race

Local-spin algorithm verification - group ME property (1)

Available shared variables:

```
s_session: int // current session established in CS
m_need: int    // session of the thread
```

In CS:

```
assert s_session == m_need;
sleep(5);
assert s_session == m_need;
```

Local-spin algorithm verification - group ME property (2)

iterations=1; threads =3; sessions = 2

Search	Time	Memory	States	Result
DFS	5:20:22	1648M	43905194	Completed: No errors detected

- Local-spin algorithm satisfies group ME property

Local-spin algorithm verification - lock

- Comment out lock acquire() and release()

iterations=1; threads =3; sessions = 2

Search	Time	Memory	States	Result
DFS	0:11:34	22M	220	NOT completed: out of memory No errors detected
BFS	0:00:19	220M	44510	Assertion error

➢ Lock is essential to ensure group ME

Space-efficient FCFS algorithm verification - group ME property (1)

Available shared variables:

m_need: int // session of the thread

Added shared variables (used only in CS):

s_session: int // current session established in CS

s_num: int // number of threads in CS

s_lock: new Semaphore (1)

Space-efficient FCFS algorithm verification - group ME property (2)

In CS:

```
s_lock.acquire();
s_num++;
If (s_num == 1)
    s_session = m_need;
s_lock.release();
assert s_session == m_need;
sleep(5);
assert s_session == m_need;
s_lock.acquire();
s_num--;
s_lock.release();
```

Space-efficient FCFS algorithm verification - group ME property (3)

iterations=1; threads =3; sessions = 2

Search	Time	Memory	States	Result
DFS	8:17:17	2030M	65681631	NOT completed: out of memory No errors detected
BFS	0:16:18	2408M	1253171	NOT completed: out of memory No errors detected

- For DFS, JPF created more states to verify FCFS algorithm (local-spin algorithm: 43905194 states).
- All the results we obtained don’t show this algorithm violates group ME property.

Space-efficient FCFS algorithm verification - deadlock (1)

- FCFS part review
  - FCFS property: *i* would block on *j* if *j* completes the doorway before *i*.
  - How does it ensure FCFS property?

    turn variable vs. its local copy *turn_snap*

1. Doorway starts - *i* reads all other processes' *turn* and make a local copy of them *turn_snap*
2. *i* possibly increments its *turn* – doorway ends
3. *i* checks *turn[j]* ?= *turn_snap[j]*

Space-efficient FCFS algorithm verification - deadlock (2)

- A possible deadlock could occur when:
  1. A fast process *j*, requesting same session as a slow process *i*, enters CS repeatedly, each time over-passing *i* in the doorway and increments its *turn* variable;
  2. *i* falls asleep after exiting the doorway;
  3. The over-passing happens enough times, *turn[j]* wraps back to the value *i* read.
  4. The *i* wakes up and *j* then requests a different session.

Space-efficient FCFS algorithm verification - deadlock (3)

Random Sleep

$\texttt{session}_i = \mathit{mysession}$

for $j = 1$ **to** N **do** $\mathit{turn\_snap}[j] = \texttt{turn}_j$

~~**if** $\texttt{conflict}(\mathit{mysession})$~~

$\quad \texttt{turn}_i = (\texttt{turn}_i + 1) \bmod \cancel{12}$

for $j = 1$ **to** N **do**

$\quad$ **wait until** $(\texttt{session}_j \in \{0, \mathit{mysession}\}) \lor (\mathit{turn\_snap}[j] \neq \texttt{turn}_j)$

doorway

2

iterations=3; threads =2; sessions: first 2 iterations=1, last iteration=2

Search	Time	Memory	States	Result
DFS	0:02:23	160M	559253	Completed: No errors detected
BFS	0:03:17	666M	559253	Completed: No errors detected

➢ Though the deadlock is easily produced by java, jpf seems can't detect such deadlocks directly.

Space-efficient FCFS algorithm verification - data race

- *turn* variable clearly has race: FCFS property is ensured by checking the order of *turn* read and increment.
- JPF can detect the race:

```
===========================================
gov.nasa.jpf.tools.PreciseRaceDetector
race for: "int FcfsGME.m_turn“
Thread-0 at FcfsGME.fcfs(FcfsGME.java:69)
            "(FcfsGME.java:69)"  : putfield
Thread-1 at FcfsGME.getTurn(FcfsGME.java:24)
            "(FcfsGME.java:24)"  : getfield
===========================================
```

Questions?