Concurrent Programming
Aug-Nov 2015

LECTURE 4

MORE ON MUTUAL EXCLUSION

MADHAVAN MUKUND, S P SURESH
CHENNAI MATHEMATICAL INSTITUTE

The original Bakery

- lock() -- for thread i
```
{
  choosing[i] = 1;
  number[i] = 1 + max(number[0], ..., number[N-1]);
  choosing[i] = 0;
  for (j = 0; j < N; j++) {
    while (choosing[j]);                              -- L2
    while (number[j] != 0 &&                          -- L3
           (number[j], j) < (number[i], i));
  }
}
```

- unlock() -- for thread i
```
{
  number[i] = 0;
}
```

Using the Bakery lock

- ```
  while (1) {
       lock();
       <critical section>
       unlock();
       <remainder section>
  }
  ```
- Threads are allowed to fail or be blocked forever in the remainder section
  ```

Correctness

- Thread i is in the **doorway** while `choosing[i] = 1`
- Thread i is in the **bakery** from the time it sets `choosing[i]` to `0` till it exits the critical section
- If threads i and k are in the bakery and i entered the bakery before entered the doorway, then `number[i] < number[k]`

Correctness ...

- If thread i is in the cs and thread k in the bakery, then (`number[i],i) < (number[k],k)`
- t2 - last time i read `choosing[k]` in loop `L2`
- t3 - last time i read `number[k]` in loop L3.
- t2 < t3

Correctness ...

- tw - time when k wrote the current value of `number[k]`
- t0, t1 - times k entered and left the corresponding doorway.

  $t0 < tw < t1$.

- Two cases:
  - t2 < t0 -- i is in the bakery before k is in the doorway. `number[i] < number[k]`
  - tw < t1 < t2 < t3 -- Thread i read the latest value of `number[k]`. So `(number[i],i) < (number[k],k)`

Progress

- If no thread is in cs, and at least one thread is in the bakery, some thread reaches the cs.
- The one with the least label.

Shared registers

- All variables are MRSW registers
- Very weak assumptions required about simultaneous read and write to same variable
- **Safe register:** If a read and write overlap, the read will return any legal value
- Overlap on choosing[i]? -- Binary, alternating and essentially atomic

Shared registers: number[i]

- i writes number[i] while k reads number[i] to determine number[k]:
- Suppose the new value of number[i] is m. Its previous value is 0. The max of the other number values is m-1. So number[k] will be at least m.
- i writes number[i] while k compares labels. Clearly number[k] < number[i], so there is no danger of violation of mutual exclusion
- No danger of deadlock either, as eventual all number values stabilize

Deadlock-free mutual exclusion

- flag[0..n-1] - MRSW boolean array, initially all 0
- lock() -- for thread i

```
{
    while (exists j < i: flag[j]) {  --- entry loop
        flag[i] = 0;
        while (exists j < i: flag[j]) ; --- subentry loop
        flag[i] = 1;
    }
    while (exists j: i < j < n, flag[j]) ; --- gateway loop
}
```

- unlock() -- for thread i

```
{
  flag[i] = 0;
}
```

Lower bound

- Any algorithm ensuring mutual exclusion and global progress for n threads requires n shared variables
- Assumptions: each thread loops through **trying**, **cs**, **exit**, and **remainder** sections
- Threads can block or fail in their **remainder** sections
- Easy bounds for MRSW registers

Lower bound ...

- Consider two threads A and B and one shared register R
- Schedule A and B step by step to take them to their remainders (is this possible?)
- Run B till it is about to write to R for the first time (will it write?)
- Pause B and run A till it enters cs (why will it?)
- Resume B. It will overwrite R and enter cs. Contradiction.

Lower bound ...

- For every k <= n, and every quiescent configuration, there is a history involving only threads 0 to k-1, such that all their writes (since their last remainder section) have been obliterated and they are about to write to a distinct shared register.
- Trivial for k = 1
- Nontrivial extension from k to k+1. On the board.

Conclusion

* Mutual exclusion rocks!