Test and Set

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 10

Test and set

- The fundamental issue preventing consistent concurrent updates of shared varuables is `test-and-set`

Test and set

- The fundamental issue preventing consistent concurrent updates of shared varuables is `test-and-set`
- To increment a counter, check its current value, then add 1

Test and set

- The fundamental issue preventing consistent concurrent updates of shared varuables is `test-and-set`
- To increment a counter, check its current value, then add 1
- If more than one thread does this in parallel, updates may overlap and get lost

Test and set

- The fundamental issue preventing consistent concurrent updates of shared varuables is `test-and-set`
- To increment a counter, check its current value, then add 1
- If more than one thread does this in parallel, updates may overlap and get lost
- Need to combine test and set into an atomic, indivisible step

Test and set

- The fundamental issue preventing consistent concurrent updates of shared varuables is `test-and-set`
- To increment a counter, check its current value, then add 1
- If more than one thread does this in parallel, updates may overlap and get lost
- Need to combine test and set into an atomic, indivisible step
- Cannot be guaranteed without adding this as a language primitive

Semaphores

- Programming language support for mutual exclusion

Semaphores

- Programming language support for mutual exclusion
- Dijkstra's semaphores
  - Integer variable with atomic test-and-set operation

Semaphores

- Programming language support for mutual exclusion
- Dijkstra's semaphores
  - Integer variable with atomic test-and-set operation
- A semaphore `S` supports two atomic operations
  - `P(s)` — from Dutch passeren, to pass
  - `V(s)` — from Dutch vrygeven, to release

Semaphores

- Programming language support for mutual exclusion
- Dijkstra's semaphores
  - Integer variable with atomic test-and-set operation
- A semaphore `S` supports two atomic operations
  - `P(s)` — from Dutch passeren, to pass
  - `V(s)` — from Dutch vrygeven, to release

- `P(S)` atomically executes the following

```
if (S > 0)
  decrement S;
else
  wait for S to become positive;
```

Semaphores

- Programming language support for mutual exclusion
- Dijkstra's semaphores
  - Integer variable with atomic test-and-set operation
- A semaphore `S` supports two atomic operations
  - `P(s)` — from Dutch passeren, to pass
  - `V(s)` — from Dutch vrygeven, to release

- `P(S)` atomically executes the following

```
if (S > 0)
  decrement S;
else
  wait for S to become positive;
```

- `V(S)` atomically executes the following

```
if (there are threads waiting
    for S to become positive)
  wake one of them up;
    //choice is nondeterministic
else
  increment S;
```

Using semaphores

- Mutual exclusion using semaphores

```
Thread 1                          Thread 2
...                               ...
P(S);                             P(S);
// Enter critical section         // Enter critical section
   ...                               ...
// Leave critical section         // Leave critical section
V(S);                             V(S);
...                               ...
```

Using semaphores

- Mutual exclusion using semaphores

```
Thread 1                    Thread 2
...                         ...
P(S);                       P(S);
// Enter critical section   // Enter critical section
   ...                         ...
// Leave critical section   // Leave critical section
V(S);                       V(S);
...                         ...
```

- Semaphores guarantee
  - Mutual exclusion
  - Freedom from starvation
  - Freedom from deadlock

Problems with semaphores

- Too low level

Problems with semaphores

- Too low level
- No clear relationship between a semaphore and the critical region that it protects

Problems with semaphores

- Too low level
- No clear relationship between a semaphore and the critical region that it protects
- All threads must cooperate to correctly reset semaphore

Problems with semaphores

- Too low level
- No clear relationship between a semaphore and the critical region that it protects
- All threads must cooperate to correctly reset semaphore
- Cannot enforce that each `P(S)` has a matching `V(S)`

Problems with semaphores

- Too low level
- No clear relationship between a semaphore and the critical region that it protects
- All threads must cooperate to correctly reset semaphore
- Cannot enforce that each `P(S)` has a matching `V(S)`
- Can even execute `V(S)` without having done `P(S)`

Summary

- Test-and-set is at the heart of most race conditions
- Need a high level primitive for atomic test-and-set in the programming language
- Semaphores provide one such solution
- Solutions based on test-and-set are low level and prone to programming errors