Sistemas distribuidos
Comunicación por memoria compartida

Sergio Yovine

Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina

Sistemas Operativos, segundo cuatrimestre de 2015

(2) Exclusión mutua: Locks

- Objeto atómico básico: *get-and-set*

```
1  atomic bool get() {
2    return mutex;
3  }
4
5  atomic void set(bool b) {
6    mutex = b;
7  }
8
9  atomic boolean getAndSet(bool b) {
10   bool m = mutex;
11   mutex = b;
12   return m;
13 }
14
15 atomic boolean testAndSet() {
16   return getAndSet(true);
17 }
```

(3) Exclusión mutua: Locks

- Spin lock (TASLock)

```
1  void create() {
2    mutex.set(false);
3  }
4
5  void lock() {
6    while (mutex.testAndSet()) {}
7  }
8
9  void unlock() {
10   mutex.set(false);
11 }
```

(4) Exclusión mutua: Locks

- Spin lock (TTASLock)

```
1  void create() {
2    mutex.set(false);
3  }
4
5  void lock() {
6    while (true) {
7      while (mutex.get()) {}
8      if (!mutex.testAndSet()) return;
9    }
10 }
11
12 void unlock() {
13   mutex.set(false);
14 }
```

(5) Exclusión mutua: TASLock vs TTASLock

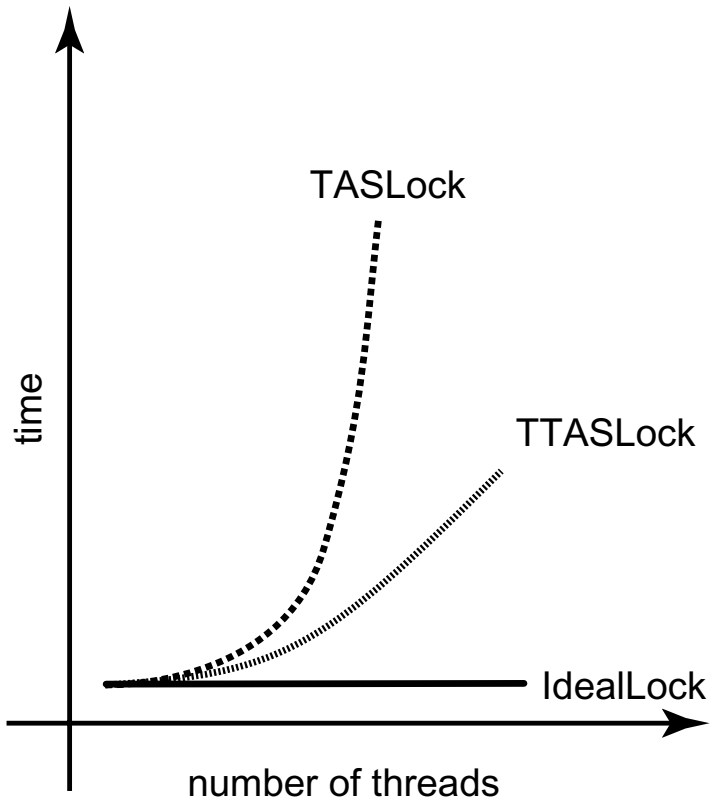


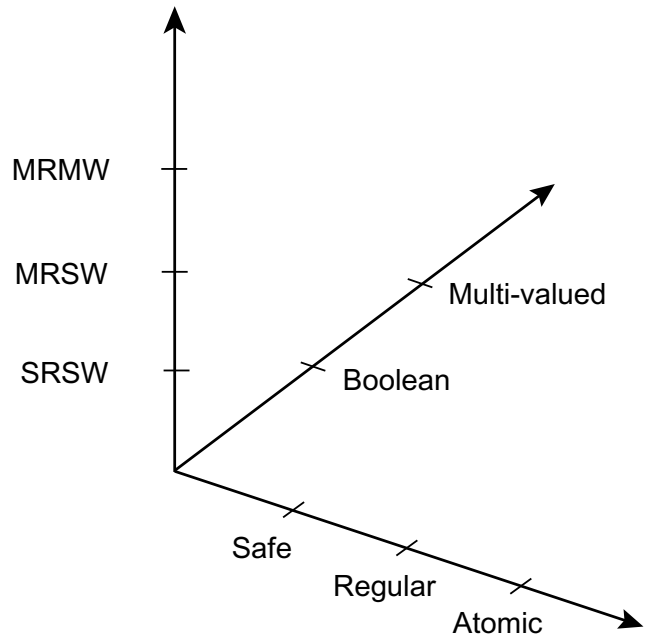
Figure 7.4 Schematic performance of a TASLock, a TTASLock, and an ideal lock with no overhead.

(6) Exclusión mutua sin locks

¿ Es posible garantizar exclusión muta sin TAS ?

Objeto atómico básico:
read-write register

Procesos por operación:
single/multiple



`read()` devuelve último valor escrito, pero
si `read()` y `write()` se solapan

- Safe: `read()` devuelve cualquier valor
- Regular: `read()` devuelve algún valor escrito

Atomic: no hay solapamiento

(7) Exclusión mutua sin locks: Dijkstra

- Registros
  - flag[i]: atomic single-writer / multi-reader
  - turn: atomic multi-writer / multi-reader
- Process *i*

```
1  /* Try */
2  L: flag[i] = 1;
3  while (turn ≠ i)
4     if (flag[turn] == 0) turn = i;
5  flag[i] = 2;
6  foreach j ≠ i
7     if (flag[j] == 2) goto L;
8  /* Crit */
9  ...
10 /* Exit */
11 flag[i] = 0;
```

- Garantiza EXCL y PROG, pero no G-PROG 

(8) Exclusión mutua sin locks: Panadería de Lamport

- Registros
  - choosing[i], number[i]: atomic single-writer / multi-reader
- Process *i*

```
1  /* Try */
2  choosing[i] = 1;
3  number[i] = 1 + max_{j≠i} number[i];
4  choosing[i] = 0;
5  foreach j ≠ i {
6     waitfor choosing[j]==0;
7     waitfor number[j]==0 ||
8             (number[i], i) < (number[j], j);
9  }
10 /* Crit */
11 ...
12 /* Exit */
13 number[i] = 0;
```

- Garantiza EXCL, PROG y G-PROG 

(9) Exclusión mutua sin locks: Resumen

- Registros *atomic multi-writer / multi-reader*
  - EXCL y PROG, pero no G-PROG
    - Dijkstra
  - EXCL, PROG y G-PROG
    - Peterson
    - Tournament

- Registros *atomic single-writer / multi-reader*
  - EXCL y PROG, pero no G-PROG
    - Burns
  - EXCL, PROG y G-PROG
    - Lamport (Panadería)
    - Usa contadores (*time-stampping*) no acotados
    - Hay soluciones con contadores acotados

(10) Exclusión mutua sin locks: Propiedades

Complejidad

- Los algoritmos vistos requieren $\mathcal{O}(n)$ registros RW

Teorema (Burns & Lynch)

> No se puede garantizar EXCL y PROG con menos de n registros RW

¿Se puede hacer algo mejor?

- Sí, pero asumiendo restricciones de tiempo
- Algoritmo de Michael Fischer

(11) Exclusión mutua sin locks: Fischer

- Registros
  - turn: multi-writer / multi-reader
- Process i

```
1  /* Try */
2  L: waitfor turn = 0;
3  turn = i;
4  pause Δ;
5  if turn ≠ i goto L;
6  /* Crit */
7  ...
8  /* Exit */
9  turn = 0;
```

tarda a lo sumo δ

- Garantiza EXCL y PROG si $\Delta > \delta$ 

(12) Consenso

Teorema (Herlihy, Lynch)

No se puede garantizar consenso con registros RW atómicos

Jerarquía de objetos atómicos (Herlihy)

Consensus number:

Cantidad de procesos para los que resuelve consenso

- Registros RW atómicos $= 1$
- Colas, pilas $= 2$
- `getAndSet()` $= 2$

¿Existen objetos atómicos con consensus number mayor?

(13) Consenso: Compare-and-swap

- Compare-and-swap/set

```
1  atomic T compareAndSwap(T u, T v) {
2    T w = register;
3    if (u == w) register = v;
4    return w;
5  }
6
7  atomic bool compareAndSet(T u, T v) {
8    T w = register;
9    if (u == w) register = v;
10   return w == v;
11 }
```

- Teorema: `compareAndSwap()` tiene consensus number infinito
- Corolario: No se puede garantizar exclusión mutua *wait-free* con registros RW. Si fuera posible, se podría implementar `compareAndSwap()` y realizar consenso.
- `compareAndSwap()` en HW: Intel x86 cmpxchg

(14) Bibliografía extra

- M. Herlihy, N. Shavit. The Art of Multiprocessor Programming. Morgan Kaufmann, 2008.
- M. Abadi, L. Lamport. An Old-Fashioned Recipe for Real Time. ACM TOPLAS 16:5, 1994. `http://goo.gl/t0Uir8`
- M. Herlihy. Impossibility and universality results for wait-free synchronization. ACM PODC, 1988.`http://goo.gl/arpWeP`
- L. Lamport. A new solution of Dijkstra's concurrent programming problem. CACM 17:8,1974. `http://goo.gl/AZpjw0`
- N. Lynch, N. Shavit. Timing-Based Mutual Exclusion. IEEE 13th RTSS, 1992. `http://goo.gl/M1EtQD`