Acciones Atómicas complejas

Habíamos supuesto que

- Acciones atómicas:
  - Read
  - Write
- Vamos a suponer que hay otras acciones atómicas
  - Test-and-set
  - Exchage
  - Fetch-and-add
  - Compare-and-swap

Test-and-set

Test-and-set(compartida, local) es

local :=compartida;

compartida:=1;

} Se ejecutan de manera atómica sin posibilidad de interleaving

Test-and-set

- Usar una variable compartida **compartida**= 0

ThreadId = 0

```
int local
Sección no crítica
repetir
    test-and-set(compartida,local)
hasta (local = 0)
Sección crítica
compartida = 0
Sección no crítica
```

ThreadId = 1

```
int local
Sección no crítica
repetir
    test-and-set(compartida,local)
hasta (local = 0)
Sección crítica
compartida= 0
Sección no crítica
```

Exchange

exchange(a,b) es

temp = a

a := b;

b :=temp;

Se ejecutan de manera atómica sin posibilidad de interleaving

Exchange

- Usar una variable compartida **compartida** = 1

ThreadId = 0

```
int local = 0
Sección no crítica
repetir
    exchange(compartida,local)
hasta (local = 1)
Sección crítica
compartida= 1
Sección no crítica
```

ThreadId = 1

```
int local =0
Sección no crítica
repetir
    exchange(compartida,local)
hasta (local = 1)
Sección crítica
compartida= 1
Sección no crítica
```

Fetch-and-add

Fetch-and-add(compartida, local, x) es

local = compartida

compartida := compartida + x;

Se ejecutan de manera atómica sin posibilidad de interleaving

Fech-and-add

- Usar dos variables compartida
  - **ticket** = 0
  - **turno** = 0

ThreadId = 0

Int miturno
Sección no crítica
fetchAndAdd(ticket, miturno,1)
while (turno !=miturno){}
Sección crítica
Fetch-and-add(turno,miturno,1)
Sección no crítica

ThreadId = 1

Int miturno
Sección no crítica
fetchAndAdd(ticket, miturno,1)
while (turno !=miturno){}
Sección crítica
Fetch-and-add(turno, miturno,1)
Sección no crítica

Compare-and-swap

```
CAS(compartida, viejo, nuevo) es
    temp = compartida
    if compartida = viejo then
        compartida := nuevo;
    return temp;
```

Se ejecutan de manera atómica sin posibilidad de interleavings

Compare and swap

- Usar una variable compartida **compartida** = false

ThreadId = 0

```
Sección no crítica
while(!CAS(compartida,false, true))
   {}
Sección crítica
compartida= false
Sección no crítica
```

ThreadId = 1

```
Sección no crítica
while(!CAS(compartida,false, true))
   {}
Sección crítica
compartida= false
Sección no crítica
```

Busy-waiting

- Todas estas soluciones son poco eficientes:
  - Consumen tiempo de procesador sin hacer cómputo efectivo
- Una alternativa sería
  - Suspender la ejecución de un proceso que intenta entrar a la región crítica hasta tanto sea posible