

Mutua esclusione

Concetti

- **Problema della mutua esclusione:** problema che nasce quando più processi vogliono accedere a variabili (risorse) comuni.
- **Azione atomica:** esegue una trasformazione di stato indivisibile. Può esistere uno stato intermedio, ma non è rilevabile all'esterno.
 - I valori base sono memorizzati e letti in modo atomico;
 - Ciascun processo ha il proprio set di registri e i dati sono manipolati nei registri;
 - Ogni risultato intermedio è memorizzato in registri o memoria privata del processo;
- **Sezione critica:** sequenza di istruzioni con le quali un processo accede e modifica un insieme di variabili comuni;
 - **Regola di mutua esclusione:** *sezioni critiche appartenenti alla stessa classe devono escludersi mutuamente nel tempo* (oppure: una sola sezione critica di una classe può essere in esecuzione in ogni istante).
- **Schema generale:** ogni processo, prima di entrare in una sezione critica, chiede autorizzazione per l'utilizzo esclusivo della risorsa (prologo); al termine, la rilascia (epilogo).

Proprietà di una soluzione al problema della mutua esclusione

1. Sezioni critiche della stessa classe devono essere eseguite in modo mutuamente esclusivo;
2. Quando un processo è all'esterno di una sezione critica, non può rendere impossibile l'accesso alla stessa sezione ad altri processi;
3. Non deve essere possibile il verificarsi di situazioni in cui i processi impediscono mutuamente la prosecuzione della loro esecuzione (*deadlock*);
4. Se sono verificate le condizioni logiche per l'accesso alla sezione critica, il processo non può essere indefinitamente ritardato a causa della esecuzione della stessa sezione da parte di altri processi (*starvation*);
5. Devono essere eliminate le forme di attesa attiva (*busy waiting*)

Soluzioni Hardware

- Si applica nel caso multiprocessore.
- Macchine che posseggono istruzioni per esaminare e modificare una parola di memoria o scambiarne due, in un solo ciclo di memoria.

<pre>int x = 1; // Processo 1 main() { // ... lock (&x); << sezione critica >>; unlock (&x); // ... }</pre>	<pre>// Processo 2 main() { // ... lock (&x); << sezione critica >>; unlock (&x); // ... }</pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------

- E' necessario che le operazioni di lock e unlock siano **indivisibili**.
 - Esecuzione contemporanea sequenzializzata dall'hardware;
 - La starvation va eliminata con politiche di arbitraggio per l'accesso in memoria;
 - Rimane attesa attiva.

Comportamento logico di lock e unlock

<pre>void lock(int* x) { while(!*x); *x = 0; }</pre>	<pre>void unlock(int* x) { *x = 1; }</pre>
--------------------------------------------------------------	------------------------------------------------

Implementazione lock con istruzione Test&Set

<pre>void lock (int* x) { while(!TestAndSet(x)); } int TestAndSet(int* a) { int R = *a; *a = 0; return R; }</pre>	<pre>// Realizzazione TestAndSet ASM: // realizzata dall'ISA in un solo ciclo di memoria! lock(x): tsl register, x // register <- x, x <-- 0 cmp register, 1 // register == 1? jne lock // se x = 0 ricomincio ret // termino</pre>
------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Implementazione lock con istruzione Exchange

<pre>void lock (int* x) { int priv = 0; do { EXCH(x, &priv); } while (priv == 0); }</pre>	<pre>// Realizzazione Exchange: // realizzata dall'ISA in un solo ciclo di memoria! int EXCH(int* a, int* b) { int temp = *a; *a = *b; *b = temp; }</pre>
-----------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

Soluzione #1: disattivazione degli interrupt

Codice

```
main()
{
  // ...
  << disattivazione interrupt (asm CLI) >>
  << sezione critica >>
  << riattivazione interrupt (asm STI) >>
  // ...
}
```

Svantaggi

- Valida solo per un uP
- Elimina ogni possibilità di parallelismo
- Rende insensibile il sistema ad ogni stimolo esterno

Soluzione #2: Utilizzo di una variabile “libero”

Codice

<pre>int libero = 1; // Processo 1 main() { // ... while (!libero); libero = 0; << sezione critica >> libero = 1; // ... }</pre>	<pre>// Processo 2 main() { // ... while (!libero); libero = 0; << sezione critica >> libero = 1; // ... }</pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------

Svantaggi

- Non risolve il problema della mutua esclusione

Soluzione #3: Utilizzo di una variabile “turno”

Codice

<pre>int turno = 1; // turno IN (1, 2) // Processo 1 main() { // ... while (turno != 1); << sezione critica >>; turno = 2; // ... }</pre>	<pre>// Processo 2 main() { // ... while (turno != 2); << sezione critica >>; turno = 1; // ... }</pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------

Svantaggi

- Impone un vincolo di alternanza non richiesto

Soluzione #4: Due variabili “libero”

Codice

```
int libero1 = 0;
```

<code>int libero2 = 0;</code>	
<pre>// Processo 1 main() { // ... libero1 = 1; while (!libero2); << sezione critica >> libero1 = 0; // ... }</pre>	<pre>// Processo 2 main() { // ... libero2 = 1; while (!libero1); << sezione critica >> libero2 = 0; // ... }</pre>

Vantaggi

- Risolve la mutua esclusione

Svantaggi

- Pone un problema di possibile deadlock se si eseguono subito le due assegnazioni

Soluzione #5: Controllo dell'altro processo

Codice

<code>int libero1 = 0;</code>	
<code>int libero2 = 0;</code>	
<pre>// Processo 1 main() { // ... libero1 = 1; while (libero2) { libero1 = 0; while (libero2); libero1 = 1; } << sezione critica >> libero1 = 0; // ... }</pre>	<pre>// Processo 2 main() { // ... libero2 = 1; while (libero1) { libero2 = 0; while (libero1); libero2 = 1; } << sezione critica >> libero2 = 0; // ... }</pre>

Svantaggi

- P1 analizza libero2: se vale 1, P2 è nel suo prologo, quindi P1 resetta il suo stato e aspetta; non risolve il problema del deadlock, basta che i processi partano insieme e procedano alla stessa velocità.

Soluzione #6: Algoritmo di Dekker (1965)

Codice

<pre>int libero1 = 0; int libero2 = 0; int turno = 1; // turno IN (1, 2)</pre>	
<pre>// Processo 1 main() { // ... libero1 = 1; while (libero2) { if (turno == 2) { libero1 = 0; while (turno != 1); libero1 = 1; } } << sezione critica >> turno = 2; libero1 = 0; // ... }</pre>	<pre>// Processo 2 main() { // ... libero2 = 1; while (libero1) { if (turno == 1) { libero2 = 0; while (turno != 2); libero2 = 1; } } << sezione critica >> turno = 1; libero2 = 0; // ... }</pre>

Svantaggi

- Se esteso a N processi, è possibile un problema di starvation: se ogni processo ha una struttura ciclica, la sezione critica viene sempre eseguita da processi con priorità maggiore (valore di turno).

Soluzione #7: Algoritmo di Peterson (1981)

Codice

<pre>int libero1 = 0; int libero2 = 0; int turno = 1; // turno IN (1, 2)</pre>	
<pre>// Processo 1 main() { // ... libero1 = 1; turno = 2; while (libero2 && turno == 2); << sezione critica >> libero1 = 0; // ... }</pre>	<pre>// Processo 2 main() { // ... libero2 = 1; turno = 1; while (libero1 && turno == 1); << sezione critica >> libero2 = 0; // ... }</pre>

Caratteristiche

- Più semplice di Dekker
- Elimina la starvation