UD 5 - SINCRONIZZAZIONE DEI PROCESSI

La compresenza in memoria di processi cooperanti implica la loro esecuzione (sincrona o asincrona) con conseguente condivisione di dati. La consistenza del dato è fondamentale per la sua corretta esecuzione.

Quando più processi accedono contemporaneamente alla stessa risorsa fisica o logica, si ha la concorrenza. Se l'esito delle operazioni dipende dall'ordine in cui sono eseguite, si ha una corsa critica, e ciò può causare inconsistenza nella risorsa.

Una porzione di codice che causa corse critiche se eseguita in modo concorrente è detta sezione critica. Ciò va evitato con meccanismi e politiche di sincronizzazione (necessaria e importantissima nei sistemi operativi).

Lo scopo dell'individuazione della sezione critica è quello di creare un protocollo che possa essere utilizzato per bypassare i problemi di inconsistenza. La soluzione deve dunque soddisfare tre condizioni:

- **mutua esclusione**: se un processo sta eseguendo la sua sezione critica e nessun altro può farlo
- **progresso**: la competizione per entrare nella sezione critica deve avvenire tra i processi che non sono nella loro sezione critica e la competizione non deve durare indefinitamente
- **attesa limitata**: per prevenire la starvation di un processo viene prefissato un numero di volte massimo secondo cui un determinato processo entra nella sezione critica dopo che un secondo ha fatto richiesta di entrare nella propria. In altre parole, un processo non deve essere costretto ad attendere indefinitamente per entrare nella sua sezione critica

La sincronizzazione deve garantire i tre punti elencati sopra attivando sistemi misti hardware/software. Ci sono diversi metodi per garantire la corretta esecuzione della sezione critica.

Variabili di turno e di lock

Un tipo di approccio a livello di istruzioni per la sincronizzazione di due processi concorrenti, sono le variabili di turno. Esse sono variabili condivise tra i processi che interagiscono per accedere in modo concorrente a una risorsa, stabilendone il turno d'uso. In altre parole dicono quale processo ha il diritto di usare una determinata risorsa in un certo istante.

Un altro approccio a livello di istruzioni sono le variabili di lock. E' una variabile condivisa che definisce lo stato di uso della risorsa, cioè quando è in uso da parte di un processo (che è evidentemente nella sua sezione critica). Cambia così il punto di vista rispetto alla variabile di turno: non sono più i processi ad alternarsi ma è la risorsa stessa a dire se è disponibile o no. Può assumere due valori: 0 se la risorsa è libera, 1 se è in uso.

Quando un processo vuole utilizzare la risorsa, deve ottenere un lock in questo modo:

- Disabilitare gli interrupt
- Leggere la variabile di lock
- Se è a 0, la imposta ad 1, riabilita gli interrupt e usa la risorsa, altrimenti riabilita gli interrupt e si mette in attesa.

La disabilitazione/abilitazione degli interrupt rende la sequenza atomica, evitando una corsa critica se più processi tentassero di ottenere il lock contemporaneamente. Molti processori forniscono un supporto hardware, l'istruzione TEST-AND-SET, che esegue proprio questa sequenza di istruzioni in un'istruzione hardware, che per definizione è indivisibile. Al rilascio, il lock viene reimpostato a 0.

Semafori

Un approccio diverso alla sincronizzazione si può ottenere con l'utilizzo dei semafori. I semafori portano la gestione della sincronizzazione a livello di sistema operativo e non più di codice come in precedenza. I semafori possono essere:

1. **Binari**: funziona come una variabile di lock al contrario. Se il valore è 1, la risorsa è libera e se è 0, la risorsa è in uso. Il semaforo viene pilotato da due funzioni atomiche (sono procedure del sistema operativo) dette acquire() e release():
    a. **Acquire**(S), usata per richiedere una risorsa: controlla il valore e se è 0 mette in attesa il processo, altrimenti lo imposta a 0 e il processo può usare la risorsa
    b. **Release**(S), usata per rilasciare la risorsa: reimposta il valore a 1
2. **Generalizzati**: è una variabile intera che rappresenta lo stato d'uso di un insieme di risorse condivise e dello stesso tipo. L'intero indica il numero di risorse libere (e dello stesso tipo), se è uguale a 0 non ci sono risorse di quel tipo utilizzabili in quel momento.

La Gestione della sincronizzazione attraverso i semafori è sicura e garantisce un uso corretto delle risorse da parte dei programmi purché il programmatore abbia inserito le corrette chiamate alle funzioni di sistema (perché i semafori tali sono).

Un possibile problema riguarda il modo con cui un processo si mette in attesa può essere un ciclo infinito che controlla il valore del semaforo (busy wait). Sebbene questo sia semplice da realizzare, in caso di attese lunghe si spreca il tempo del processore inutilmente.

Si può modificare l'operazione acquire per far si che inserisca i processi che devono attendere in una coda associata al semaforo (che può essere schedulata, ponendo attenzione alla starvation), e la release per far si che ne estragga uno dalla coda (se ve ne sono). In questo modo, non si elimina del tutto il problema del busy wait (l'accesso alla coda infatti va sincronizzato), ma lo si minimizza, poiché se anche bisogna attendere per inserire un processo in coda perché un altro processo sta venendo inserito, l'attesa durerà poco poiché si tratta di poche istruzioni. Inoltre con i semafori si possono verificare situazioni di attesa circolare (deadlock).

Il problema più grave però è che il loro corretto utilizzo spetta al programmatore, il quale può utilizzarli erroneamente o non utilizzarli affatto, e se ciò accade, ci possono essere violazioni di mutua esclusione, o si possono verificare fenomeni di starvation.

Monitor

Un altro approccio è l'utilizzo di monitor: è un sistema che previene gli errori di programmazione che, nonostante l' uso dei semafori, non siano trattabili dal sistema (e quindi sfuggono alla gestione interna del sistema operativo).

Il motivo per cui il controllo potrebbe sfuggire al sistema operativo è che le chiamate alle funzioni dei semafori potrebbero essere fatte in maniera non corretta. Il programmatore potrebbe dimenticarsi o chiamare (acquire) la funzione nel momento sbagliato o dimenticarsi di eseguire il release di una risorsa (in questo caso il risultato sarebbero attese infinite in altri processi).

Il monitor è un sistema che favorisce la sincronizzazione garantendo l'esecuzione delle chiamate necessarie alle funzioni di sistema operativo per l'utilizzo delle risorse. Si tratta di un tipo di dato astratto, che, oltre a dei dati, possiede alcuni metodi che sono eseguiti in mutua esclusione. Questa però è gestita a livello di linguaggio di programmazione, in particolare dal compilatore che farà uso delle corrette chiamate di sistema, e quindi non spetta più al programmatore il loro corretto utilizzo.