**Race condition?**

Sono situazioni in cui più processi accedono e modificano gli stessi dati in modo concorrente e i risultati dipendono dall’ordine degli accessi.

**Che cos’è la sezione critica e come si risolve?**

La sezione critica è un segmento di codice appartenente ad un processo, nella quale è possibile modificare delle variabili che sono visibili e modificabili anche dagli altri processi.

Se un processo è nella sezione critica nessun altro processo può entrare nella propria sezione critica, quindi l’esecuzione della sezione critica da parte dei processi è mutuamente esclusiva.

Una soluzione alla sezione critica deve rispettare 3 requisiti deve garantire la mutua esclusione, il progresso e l’attesa limitata.

Mutua esclusione se un processo è in esecuzione nella propria sezione critica nessun altro processo può essere contemporaneamente in esecuzione nella propria sezione critica.

Progresso Se la sezione critica è libera ed esiste qualche processo che vuole entrare non si può impedire per alcun motivo l’accesso di quel processo alla sezione critica, se un processo vuole entrare nella propria sezione critica deve entrare e basta, questo vieta di utilizzare algoritmi a rotazione statica.

Attesa limitata Se un processo vuole accedere alla sezione critica esiste un numero di volte che consente ad altri processi di entrare in sezione critica prima o poi toccherà a lui.

**Algoritmo di Peterson?**

L’algoritmo di Peterson è una soluzione software al problema della sezione critica e si applica soltanto a due processi.

La soluzione di Peterson prevede che i due processi condividono due dati:

1. Una variabile turno che indica di chi è il turno di entrare alla sezione critica.
2. Un array flag con quale si manifesta la volontà di un processo di entrare nella sezione critica.

L’algoritmo di Peterson garantisce tutti e tre i requisiti.

Quando un processo vuole entrare nella sezione critica per prima cosa imposta il valore di flag a true dopodiché assegna il turno all’altro processo conferendo così il turno all’altro processo.

Se i due processi tentano l’accesso contemporaneo alla sezione critica entra il primo processo che ha manifestato la volontà di entrare perché il secondo ha assegnato il turno al primo.

**Che cos’è un semaforo?**

Un semaforo è una variabile intera alla quale si può accedere solo attraverso due operazione atomiche predefinite e sono la wait (decrementa il valore della variabile) e signal (aumenta il valore della variabile).

La wait e la signal devo essere eseguite in maniera indivisibile mentre un processo cambia il valore del semaforo nessun altro processo può contemporaneamente modificare quello stesso valore.

Esistono due tipologie di semafori, i semafori contatori il cui valore numerico è illimitato e i semafori binari il cui valore è compreso tra 0 ed 1.

Se il semaforo vale 0 ed un processo richiede una risorsa deve attendere che il valore del semaforo ritorni positivo.

I semafori sono utilizzabili per risolvere il problema della sezione critica ad n processi, gli n processi condividono un semaforo in comune chiamato mutex inizializzato ad 1.

Il principale svantaggio di un semaforo è che richiede sempre un’attesa attiva (busy waiting) ovvero se un processo tenta di accedere alla propria sezione critica ma un altro processo si trova in sezione critica il processo che tenta l’accesso resterà nel ciclo del codice d’ingresso.

Questa soluzione costituisce un problema per i sistemi multiprogrammazione perché si ha uno spreco di cicli di CPU.

Una soluzione all’attesa attiva consiste in una ridefinizione dell’operazioni wait e signal.

Se un processo tenta di accedere alla sezione critica me questa è già occupata invece di rimanere a loopare nella sezione di codice di ingresso e quindi rimane in attesa viene bloccato ovvero il processo passa in stato d’attesa e viene aggiunti alla coda d’attesa associata al semaforo. Cosi la CPU è libera e può schedulare un altro processo.

Un processo bloccato che attende un semaforo verrà riavviato inseguito ad una signal effettuata da un altro processo sullo stesso semaforo. Il processo viene riavviato tramite un’operazione detta wakeup, in questo caso lo stato del processo passa da attesa a pronto ed il processo entra nella coda dei processi pronti.

Viene risvegliato un processo qualsiasi di quelli presenti nella lista.

**Che significa in maniera atomica?**

Un’istruzione è detta atomica se la sua esecuzione non è sovrapponibile all’esecuzione della stessa su di un'altra CPU.

**Che cos’è un monitor?**

Un monitor è un tipo di dato astratto che al suo interno contiene sia strutture dati e sia procedure.

Il monitor di per sé assicura che quando si lavora su queste strutture dati che sono i dati shared che si trovano all’interno di esso, sì lavori a uno per volta ossia il programmatore non deve preoccuparsi di garantire la mutua esclusione perché la mutua esclusione e garantita dal fatto che già si è al interno del monitor.

Quindi un monitor contiene sia i dati shared e sia le operazioni che vogliamo fare sui dati shared più il codice di inizializzazione e nel monitor si entra a uno per volta.

La sincronizzazione su condizione cioè il fatto di potersi fermare quando una certa condizione è non valida va opportunamente programmato, per fare questo i costrutti di tipo monitor prevedono l’utilizzo di particolari variabili che si chiamano variabili conditional che servono per la sincronizzazione su condizione.

Si tratta di particolari variabili su cui sono definite due operazioni che si chiamano wait e signal (non sono quelle dei semafori).

Se un processo fa un wait su di una variabile conditional viene messo in una coda di attesa gestita in maniera FIFO.

Signal sveglia il processo che sta dormendo da più tempo se è presente.

Nella struttura del monitor ci sono le variabili conditional.

I processi vengono addormentati quando la condizione non è rispettata.

Solo un processo può essere attivo al interno del monitor, perché per definizione l’accesso all’interno del monitor avviene a 1 per volta.

Se un processo all’interno del monitor invoca una signal su una variabile conditional per risvegliare un processo l’altro processo non può andare in esecuzione conte

mporanea in questo caso abbiamo due possibilità:

1. Si sospende il processo che ha fatto la segnalazione e attende che il processo risvegliato esca questa tecnica si chiama segnala e attendi.
2. Si risveglia il processo ma questo non andrà in esecuzione immediatamente ma andrà in esecuzione non appena il monitor sarà vuoto e chi ha segnalato cioè quello che risveglia continua la sua esecuzione questa tecnica è detta segnale e continua.

Segnale e attendi ottima soluzione quella utilizzata di più è la seconda.

**Aggiunte Johnny**

Esistono anche delle soluzioni hardware al problema della sezione critica.

La tecnica più semplice per evitare che un processo venga interrotto nella propria sezione critica lasciando le cose a metà e consentendo l’accesso ad un altro processo in sezione critica consiste nel disabilitare le interruzioni, se le interruzioni sono disabilitate non può scattare il time ed assegnare la CPU ad un altro processo fin quando il processo non esce dalla propria sezione critica e riabilita le interruzioni.

L’applicazione di questa tecnica non è praticabile nei processori moderni perché non è lo scattare del timer a mandare in esecuzione un altro processo ma un altro processo è già in esecuzione su di un altra CPU.

Nei sistemi moderni viene utilizzato un altro meccanismo si tratta di istruzioni basate sull’idea del locking e su istruzioni non interrompibili di tipo atomico.

L’ idea è quella che prima di entrare nella sezione critica bisogna acquisire il lock (diritto d’accesso) e rilasciarlo quando si esce dalla sezione critica

Nei processori moderni è disponibile un’istruzione detta Test And Set implementata atomicamente quindi in un sistema multicore non andranno mai in esecuzione due istruzioni dello stesso tipo in contemporanea ma una prima e l’altra dopo.

La test and set accede ad una locazione di memoria prelevandone il valore e lo testa, indipendentemente da quello che c’è dentro ci scrive 1 o true.

Se c’era scritto 1 rimane 1 se c’era scritto 0 ci scrivo 1 (0 significa libero 1 occupato) con questo sistema io vado a vedere se la sezione critica e libera.

Se c’è 0 atomicamente ci scrivo 1 in modo tale che un altro troverà 1 e rimarrà bloccato e non potrà accedere alla sezione critica.

Con la test and set i primi requisiti sono garantiti.