Sincronizzazione : Competizione e collaborazione fra processi

Più processi *interferiscono* fra loro se necessitano di usufruire della stessa risorsa. Si dice che si è generata una condizione di corsa (race condition).
Il sistema operativo deve gestire questa condizione garantendo che:

- vi sia mutua esclusione nel senso che se un processo st6a utilizzando una risorsa, nessun altro processo può appropriarsene
- un processo che voglia accedere ad una risorsa debba attendere un tempo finito
- non si verifichi la condizione di lockout: si ha lockout quando un processo si blocca in attesa di una risorsa A, ma avendo il possesso di un'altra risorsa B impedisce l'avanzamento di tutti i processi che vogliono accedere alla risorsa B.

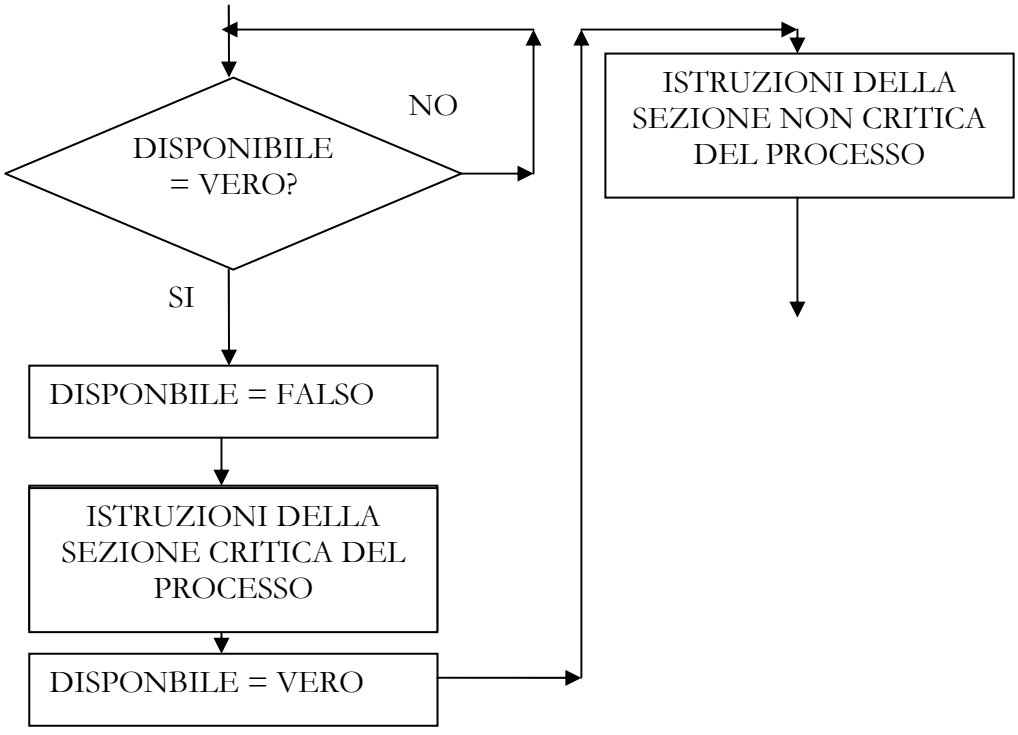
Due processi *cooperano* quando uno dei due processi, per poter avanzare necessita di una risorsa prodotta dall'altro processo.
Si parla di *condizione di stallo* quando due processi non possono avanzare se ognuno dei due necessita di una risorsa prodotta dall'altro processo per poter avanzare.

Sincronizzazione fra processi

Tornando al problema della interferenza fra processi si dice che un processo sta eseguendo una sezione critica quando sta utilizzando una risorsa con l'esclusione dell'accesso alla risorsa da parte dei tutti gli altri processi.
Esistono vari algoritmi per la gestione delle interferenze

Algoritmo 1

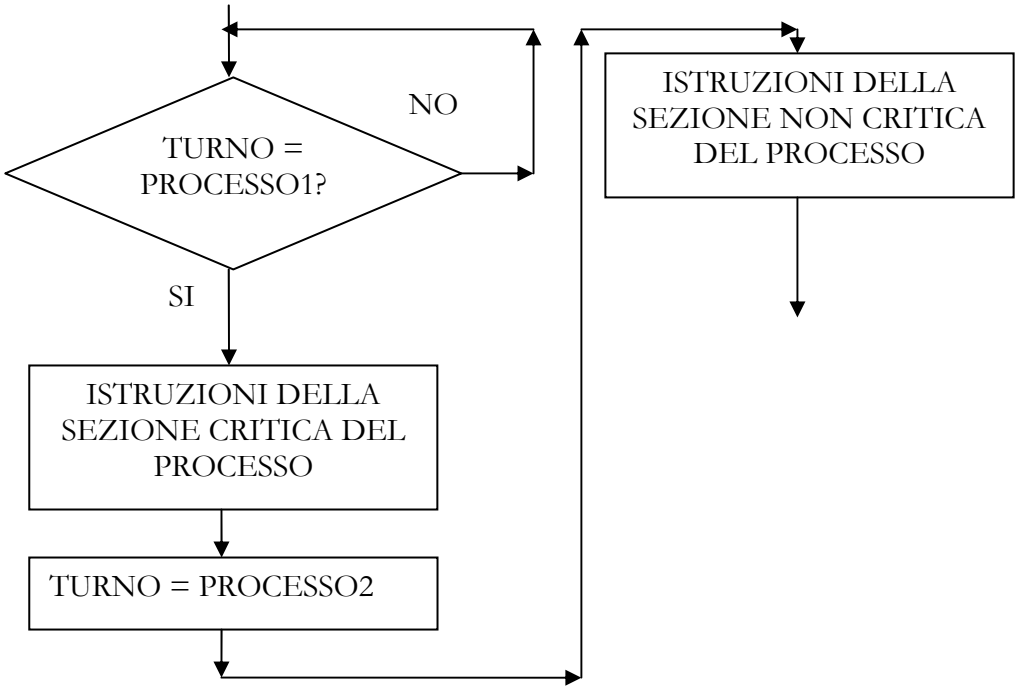
Un algoritmo molto semplice è il seguente: nel sistema esiste un avariabile a cui i processi possono accedere e modificare, una variabile che chiameremo DISPONIBILE: se tale variabile è posta a FALSO, la risorsa non è disponibile e i processi non possono accedere a tale variabile. Se essa assume il valore VERO, è disponibile per essere utilizzata da qualche processo.
Un processo per poter accedere ad una risorsa deve eseguire il seguente segmento di programma



Questo algoritmo è semplice ma non garantisce la mutua esclusione in ogni caso. Vediamo ad esempio, la seguente sequenza di eventi
La variabile DISPONIBILE viene inizializzata a VERO
Il processo P1 va in esecuzione
Il processo P1 controlla la variabile DISPONIBILE e la trova al valore VERO
Il processo P1 viene interrotto e va in esecuzione il processo P2
Il processo P2 controlla la variabile DISPONIBILE e la trova al valore VERO
Il processo P2 viene interrotto e va in esecuzione il processo P1
Il processo P1 pone la variabile DISPONIBILE a FALSO
Il processo P1 viene interrotto e va in esecuzione il processo P2
Il processo P2 pone la variabile DISPONIBILE a FALSO
Il processo P2 viene interrotto e va in esecuzione il processo P1
Il processo P1 si appropria della risorsa
Il processo P1 viene interrotto e va in esecuzione il processo P2
Il processo P2 si appropria della risorsa
Come si vede da questa sequenza di eventi, l'algoritmo, in questo caso, non è in grado di garantire la mutua esclusione.

Algoritmo 2

In questo algoritmo vi è una variabile che chiamiamo ad esempio TURNO, la quale indica quale processo può utilizzare la risorsa. Un processo, per poter utilizzare la risorsa, deve seguire il seguente programma

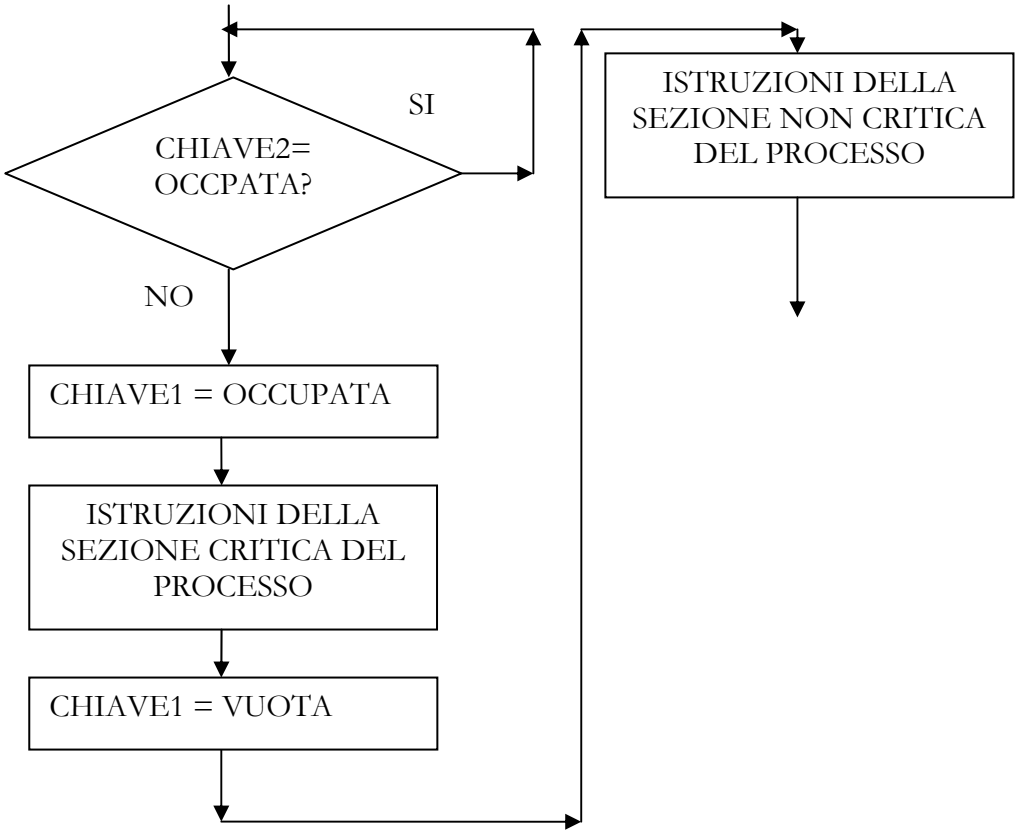


Questo algoritmo evita il problema visto nel caso precedente. Infatti un processo non può entrare nella propria sezione critica se la variabile TURNO non contiene il nome del processo stesso. Poiché il processo che detiene la risorsa non modifica il valore della variabile TURNO finché non ha terminato la sua sezione critica, il secondo processo non potrà accedere alla risorsa.
Anche questo algoritmo ha dei difetti:

- supponiamo che P1 vena eseguito raramente. Poiché il processo 2 deve aspettare che la variabile TURNO gli dia la possibilità di accedere alla risorsa ma essa viene impostata dal processo1, la conseguenza è che il processo 2 deve adeguarsi al ritmo del processo 1
- se uno dei due processi si interrompe per qualche motivo durante la sua sezione critica, non cambierà mai il valore della variabile TURNO, bloccando per sempre l'altro processo
- anche se un processo si interrompe nella sua fase non critica provocherà il blocco dell'altro processo. Infatti si possono verificare i seguenti eventi
  la variabile TURNO indica che tocca al processo 1
  il processo 1 esegue la sua sezione critica
  il processo 1 imposta la variabile TURNO in modo da passare il turno al secondo processo
  il processo 1 si blocca
  il processo 2 entra in esecuzione e testa la variabile
  poiché gli è consentito il processo 2 esegue la sua sezione critica e imposta la variabile in modo da passare il turno al processo 1
  il processo 1 è bloccato e non potrà mai cambiare il valore della variabile
  il processo 2 non potrà mai eseguire la sua sezione critica

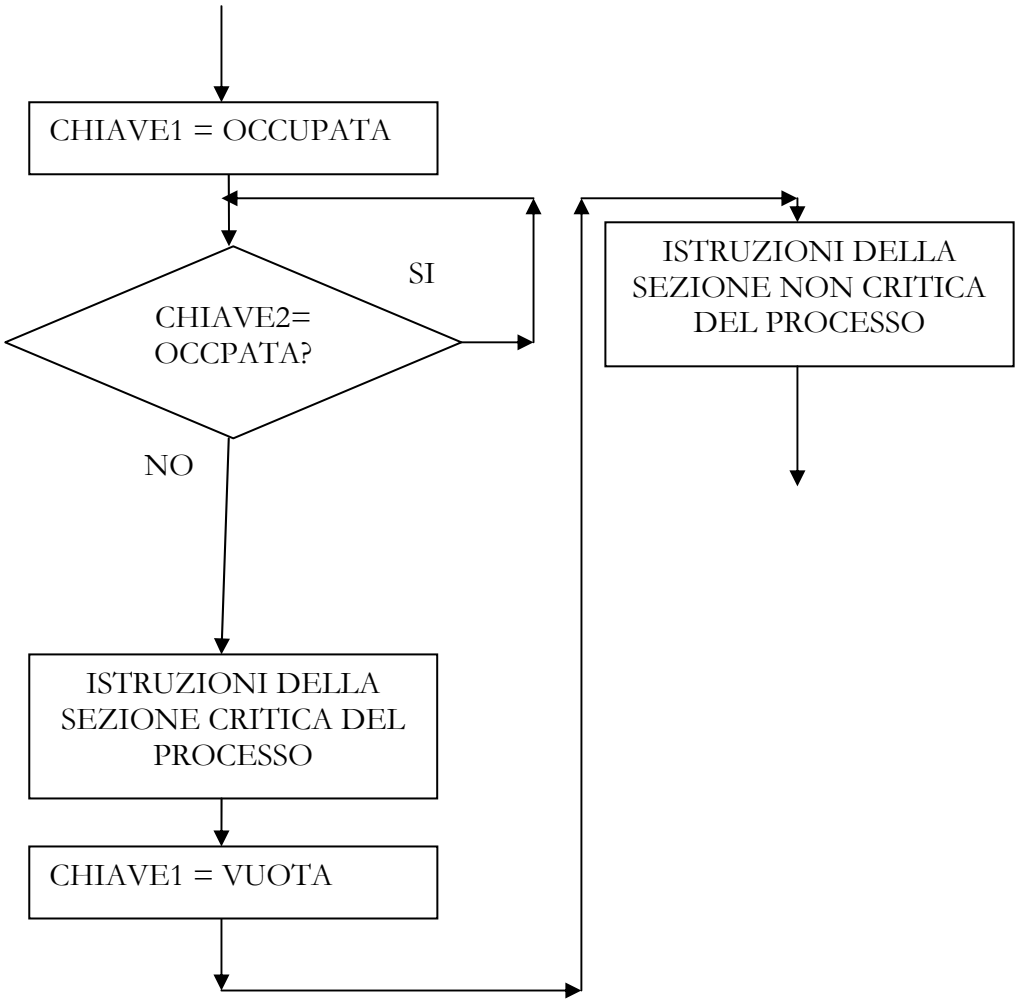
Algoritmo 3

Per risolvere il problema posto dall'algoritmo precedente, in questo nuovo algoritmo si usano due varabili diverse che chiameremo CHIAVE1 e CHIAVE2. Con la prima variabile il processo 1 indica al secondo processo se occupa la risorsa o meno. La stessa cosa fa il secondo processo con CHIAVE2. Il processo 1 può leggere la variabile CHIAVE2 ma può modificare soltanto CHIAVE1 e viceversa. Ogni processo esegue le seguenti istruzioni



L'algoritmo non impedisce che un processo, bloccandosi nella sua sezione critica, blocchi indefinitamente anche l'altro processo. Infatti se il processo 1 si blocca nella sua sezione critica non

porrà mia CHIAVE1 = VUOTA bloccando il processo 2. però se riesce a impostare tale variabile e si blocca nella sezione critica non vi è problema poiché il processo 2 troverà sempre la chiave 1 libera. Inoltre tale algoritmo non garantisce la mutua esclusione in tutti i casi. Vediamo il seguente esempio
Le variabili CHIAVE1 e CHIAVE2 vengono inizializzate al valore VUOTA
Il processo 1 va in esecuzione
Esso controlla la variabile CHIAVE 2 e la trova a VUOTA
Il processo 1 si interrompe e va in esecuzione il processo 2
Esso controlla la variabile CHIAVE 1 e la trova a VUOTA
Il processo 2 si interrompe e va in esecuzione il processo 1
Il processo 1 pone CHIAVE 1 ad OCCUPATA
Il processo 1 si interrompe e va in esecuzione il processo 2
Il processo 2 pone CHIAVE 2 ad OCCUPATA
Il processo 2 si interrompe e va in esecuzione il processo 1
Il processo 1 si appropria della risorsa
Il processo 1 si interrompe e va in esecuzione il processo 2
Il processo 2 si appropria della risorsa
Per evitare il verificarsi di questa sequenza di eventi si può modificare l'algoritmo in modo che la variabile CHIAVE venga impostata come prima azione prima di controllare l'altra chiave



Questa modifica all'algoritmo provoca però la possibilità del blocco reciproco fra i processi, come si può vedere dalla seguente sequenza di eventi

Le variabili CHIAVE1 e CHIAVE2 vengono inizializzate al valore VUOTA
Il processo 1 va in esecuzione
Il processo 1 pone CHIAVE 1 ad OCCUPATA
Il processo 1 si interrompe e va in esecuzione il processo 2
Il processo 2 pone CHIAVE 2 ad OCCUPATA
Il processo 2 si interrompe e va in esecuzione il processo 1
Il processo 1 controlla la variabile CHIAVE 2 e la trova a OCCUPATA per cui non può eseguire la sua sezione critica
Il processo 1 si interrompe e va in esecuzione il processo 2
Il processo 2 controlla la variabile CHIAVE 1 e la trova a OCCUPATA per cui non può eseguire la sua sezione critica