Il problema del produttore e del consumatore

Cooperazione tra processi

Risorsa consumabile

I processi disgiunti possono interferire tra loro a causa dell'uso di risorse permanenti, ma ognuno di essi ignora la presenza e lo scopo degli altri processi. I processi che invece interagiscono richiedendo servizi ad altri e sfruttando i risultati ricevuti, **cooperano** per il completamento di un lavoro. I processi cooperanti devono essere in grado di scambiarsi informazioni.

Nello scambio di informazioni tra processi, un processo P, detto produttore, produce e invia il *messaggio*, mentre un processo C, detto consumatore, lo riceve e lo usa. Un messaggio è cioè una risorsa **consumabile**, perché una volta acquisito non esiste più.

Variabile comune

Il problema del produttore e del consumatore

Il produttore deposita il messaggio in un'area di memoria e il consumatore lo preleva dalla stessa area.

Per determinare le regole di sincronizzazione si deve considerare che:

- il produttore può inserire un nuovo messaggio se il buffer non è pieno,
- il consumatore preleva un messaggio se il buffer non è vuoto.

Semafori

Un caso particolare del problema del produttore e del consumatore si verifica quando un processo deve essere solo avvertito che si è verificato un evento, ma non ha bisogno di alcun dato. I messaggi cioè sono vuoti, vengono solo contati. Il buffer è sostituito da un intero **v** che, in ogni istante, con il suo valore, rappresenta il numero dei messaggi inviati ma non ancora ricevuti. É una variabile di tipo semaforo.

- L'operazione 'deposita messaggio' svolta dal produttore è indicata con **signal(v)**,
- L'operazione 'preleva messaggio' svolta dal ricevitore, è indicata con **wait(v)**.

Wait su un semaforo

- La primitiva wait viene usata per ricevere le segnalazioni su un dato semaforo.
- Se il segnale non è stato inviato il processo che ha eseguito wait passa in stato di attesa.
- Se il segnale è stato inviato il processo *consuma* il semaforo e continua la sua elaborazione.

Regole di sincronizzazione

1. Sia R un intero che conta i messaggi ricevuti e S un intero che conta i messaggi spediti. Il semaforo v è la differenza tra i due contatori: v=R-S.
2. La condizione R<S si verifica quando il numero di segnali spediti è maggiore del numero di segnali ricevuti. In questa condizione, se un processo esegue **wait(v)**, allora R viene incrementata di 1 e il processo continua;
3. La condizione R=S si verifica quando tutti i segnali inviati sono stati ricevuti, in questo caso il ricevitore che esegue wait(v) viene messo in attesa, in una coda associata al semaforo v.
4. se un processo esegue **signal (v)** la variabile S viene incrementata di 1, poi, se nella coda di attesa associata al semaforo **v** vi sono processi in attesa, il sistema operativo ne seleziona uno di essi per metterlo nello stato di pronto e incrementa di 1 la variabile R.

Esempio di mutua esclusione

Le variabili di tipo semaforo consentono di realizzare l'accesso in mutua esclusione a risorse richieste da processi concorrenti.

inizialmente il semaforo v è posto a 1, le variabili S e R a 0;

Processo 1	Processo 2	. . .	Processo n
wait(v))	wait(v))		wait(v))
calcoli	calcoli		calcoli
signal(v)	signal(v)		signal(v)

Il primo processo che esegue wait(v) troverà vera la condizione R<S, e incrementerà la variabile R prima di passare alle istruzioni successive. Fino a quando questo processo non eseguirà l'operazione signal(v) ogni altro processo che tenterà di eseguire wait(v) troverà vera la condizione R=S, e verrà messo in attesa. Quindi al massimo un solo processo eseguirà la sezione *calcoli*.

Il problema del produttore e del consumatore

I requisiti della soluzione al problema del produttore e del consumatore, utilizzando le primitive di sincronizzazione wait e signal, sono:

- P e Q sono due processi in esecuzione su due CPU differenti, essi si scambiano informazioni tramite la memoria principale;
- Vi è un unico registro di scambio **r**,
  - P deve aspettare che **r** sia vuoto prima di depositarvi un nuovo messaggio;
  - Q deve aspettare che **r** sia pieno prima di prelevare un nuovo messaggio;
- Partendo da uno stato iniziale prefissato, la soluzione deve essere valida qualunque siano i tempi di esecuzione di P e di Q.

Problema del produttore e del consumatore

La soluzione impiega due semafori: FineP e FineQ.

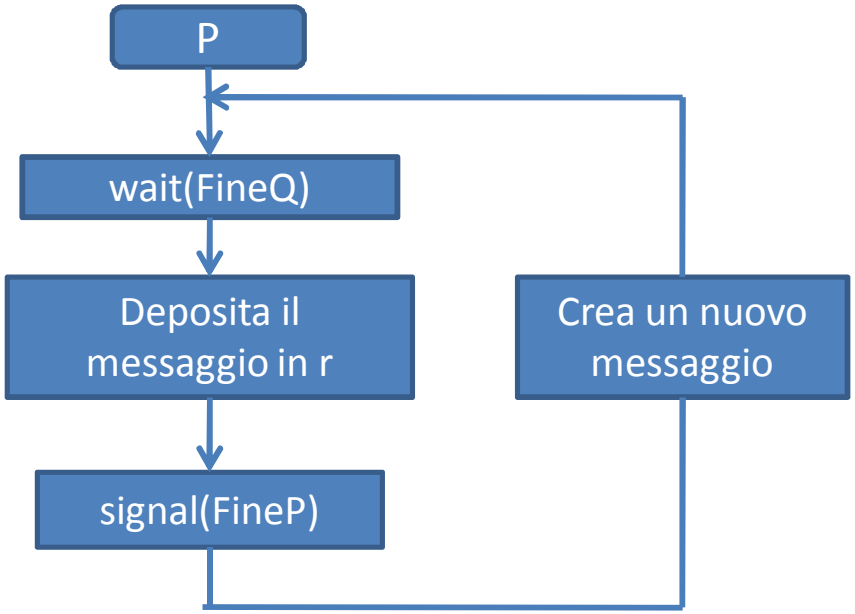
- Il semaforo FineP viene posto a vero dopo che P ha depositato un messaggio in **r**. Analogamente il semaforo FineQ è posto a vero dopo che Q termina di prelevare un messaggio da **r**.
- Il semaforo FineP è consumato dal processo Q prima del prelievo di un nuovo messaggio, il semaforo FineQ è consumato dal processo P prima del deposito di un nuovo messaggio.

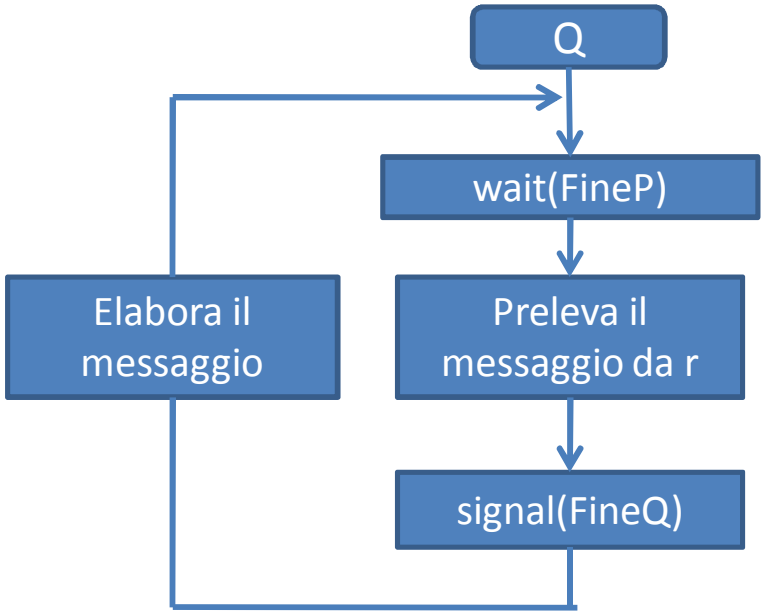
Semafori

Inizialmente:

- **r** è vuoto;
- FineP=Falso
- FineQ=vero.

Il semaforo FineP = Falso impedisce al consumatore di avanzare.
Il semaforo FineQ = Vero consente al produttore di depositare un nuovo messaggio.
I valori complementari dei semafori assicurano che in ogni istante un solo processo accede a **r**





Coda dei messaggi

Per consentire al produttore di produrre messaggi anche se il consumatore non ha ancora terminato di elaborare quello precedente, i messaggi vengono memorizzati in un array.

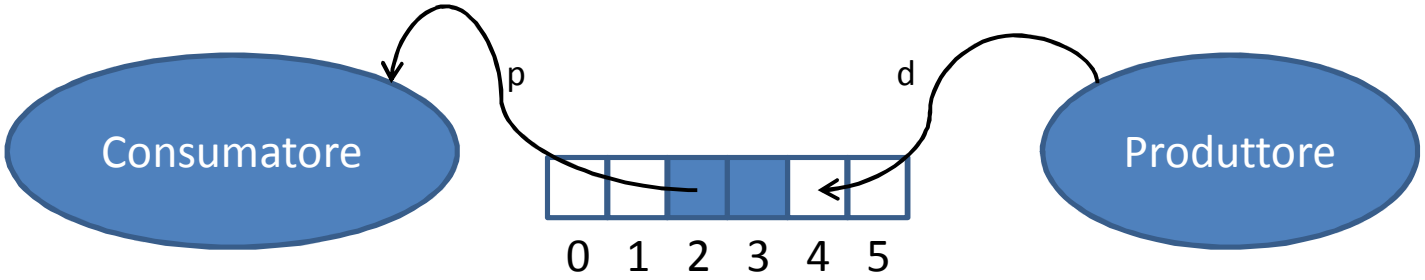
L'array viene dimensionato per contenere N record, numerati da 0 a N-1, e deve essere gestito come una coda circolare.

La variabile R è il contatore dei messaggi ricevuti, il consumatore preleva un messaggio dalla posizione:

p = R % N

La variabile S è il contatore dei messaggi spediti, il produttore deposita il nuovo messaggio nella posizione

d = S % N



Esempio:
N=6
S=22
R=20
p = 20%6=2
d = 22%6=4

Regole di sincronizzazione

Durante la comunicazione,

- il numero dei messaggi ricevuti è sempre inferiore al numero dei messaggi spediti:

$$0 \leq R \leq S.$$

- Il numero dei messaggi depositati e non ancora prelevati non può superare la capacità dell'array.

$$0 \leq S\text{-}R \leq N,$$

  che corrisponde alla condizione $S \leq R+N$.

Questa diseguaglianza, combinata con la precedente, stabilisce che, in ogni istante:

$$0 \leq R \leq S \leq R + N,$$

Le uguaglianze valgono nelle condizioni particolari di

- buffer vuoto (R=S)
- di buffer pieno (S=R+N).

Buffer di scambio dei messaggi

Il buffer è un array di N locazioni usato come una coda circolare. Vi sono m processi produttori P_1, …, P_m ed n processi consumatori Q_1, …, Q_n. La cooperazione tra i P_i e i Q_j deve soddisfare i seguenti requisiti:

- i processi P_i e Q_j sono in esecuzione su m+n CPU distinte e si scambiano messaggi tramite la memoria principale.
- Un processo P_i non può depositare in **r** un nuovo messaggio se non vi è almeno una posizione vuota;
- Un processo Q_j non può prelevare un messaggio da **r** se tutte le celle sono vuote;
- In ogni istante vi possono essere più processi P_i e Q_j che depositano e prelevano messaggi,
- La soluzione deve essere valida per qualsiasi tempo di esecuzione dei processi.

Coda circolare

- Durante l'avanzamento dei processi P_i e Q_j gli indirizzi d e p si spostano in senso circolare senza mai oltrepassarsi. Si indichi con N_v ($0 \leq N_v \leq N$) il numero di celle vuote nel buffer:

$$N_v = N - (d - p) \text{ se } d \geq p$$

Oppure

$$N_v = p - d \text{ se } d < p$$

- Per impedire ai processi P_i e Q_j di operare in situazioni anomale (array vuoto o array pieno) si usano le due primitive di sincronizzazione su un semaforo intero

Semafori di tipo intero

Tralasciando il conteggio dei messaggi spediti e dei messaggi ricevuti, il semaforo può essere usato per contare i messaggi spediti e non ancora ricevuti.

Quando un processo P_i esegue **wait(semaforo)**:

- se semaforo>0 il processo P continua la sua esecuzione e il semaforo è decrementato di uno;
- altrimenti il processo P viene posto in stato di attesa fino a quando un processo eseguirà signal(semaforo).

Quando un processo Q_j esegue la primitiva di sincronizzazione **signal(semaforo)**:

- se esiste un processo in attesa dell'evento semaforo>0 il semaforo viene lasciato a zero ma il processo viene sbloccato e passerà in esecuzione quando si rende disponibile una CPU.
- Altrimenti il semaforo viene incrementato di uno.
- In entrambi i casi il processo che ha eseguito signal(semaforo) continua l'esecuzione.

Scambio di messaggi

I processi produttori e i processi consumatori si scambiano messaggi attraverso un array gestito come coda.

Per la soluzione, Si impiegano i semafori **nvuote** (intero il cui valore indica il numero di celle vuote) e **npiene** (intero il cui valore indica il numero di celle piene nel buffer).

La variabile d è una risorsa protetta dal semaforo *dlibero* per evitare che più processi produttori depositino simultaneamente messaggi nella stessa cella del buffer. La variabile p è protetta dal semaforo *plibero* per evitare che più processi consumatori prelevino messaggi dalla stessa cella del buffer.

Inizialmente tutte le celle della coda sono vuote, cioè:

- p=1,
- d=1,
- nVuote=N,
- nPiene=0
- plibero=vero,
- dlibero=vero.

Produttori e Consumatori