Sincronizzazione dei Processi

Introduzione

Nei sistemi multi-programmati diversi thread o processi sono in esecuzione asincrona, e possono condividere dati.

I processi cooperanti possono condividere sia uno spazio di indirizzi (codice e dati), oppure solo i dati

L'accesso concorrente a dati condivisi può causare incoerenza dei dati stessi, di conseguenza è richiesto mettere in atto meccanismi atti ad assicurare un'ordinata esecuzione dei processi cooperanti

▼ Esempio: Memoria Limitata → Soluzione con memoria condivisa

Supponiamo di usare una zona di memoria condivisa e usiamo un vettore circolare di grandezza DIM, che fa uso di due indici per accedere al vettore.

Con l'indice "inserisci" puntiamo alla prima casella disponibile del vettore
Con l'indice "preleva" puntiamo alla prima posizione occupata del vettore

Al momento iniziale poniamo a zero sia inserisci che preleva.

Ricaviamo quindi che:

```
// Pseudo-codice
if inserisci == preleva // Vettore vuoto
if (inserisci + 1) % DIM == preleva // Vettore pieno
```

Esempi di codice:

```
// DATI CONDIVISI
#define BUFFER_SIZE 10
typedef struct {
  . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
int counter = 0; // Incrementata quando viene prelevato un elemento dal vettore
```

```
// PROCESSO PRODUTTORE
item nextProduced;

while (1){
  while (counter == BUFFER_SIZE); // Istruzione vuota
  buffer[in] = nextProduced;
  in = (in+1) % BUFFER_SIZE;
  counter++;
}
```

```
// PROCESSO CONSUMATORE
item nextConsumed;
while (1){
  while (counter == 0); // Istruzione vuota
  nextConsumed = buffer[out];
  out = (out+1) % BUFFER_SIZE;
  counter--;
}
```

Le istruzioni “counter++” e “counter- -” devono essere eseguite in modo atomico → Un’operazione è eseguita in modo atomico se si esegue completamente senza interruzione.

Problema della sezione critica

Si considerino n processi concorrenti:

- Ciascun processo ha un segmento di codice chiamato sezione critica, nel quale il processo può modificare variabili comuni.
- Problema → Quando un processo è in sezione critica non si deve consentire a nessun altro processo di essere in esecuzione nella propria sezione critica.

Nella gestione della sezione critica, le garanzie principali da fornire sono tre:

- Mutua esclusione
- Progresso
- Attesa limitata

Soluzione dal problema della sezione critica

Mutua Esclusione

Se il processo P_i è in esecuzione nella sua sezione critica, nessun altro processo può essere in esecuzione nella propria sezione critica.

Progresso

Se nessun processo è in esecuzione nella sua sezione critica, solo i processi che vogliono entrare nella propria sezione critica possono decidere chi saarà il prossimo a entrare, e tale decisione non può essere ritardata indefinitamente.

Attesa Limitata

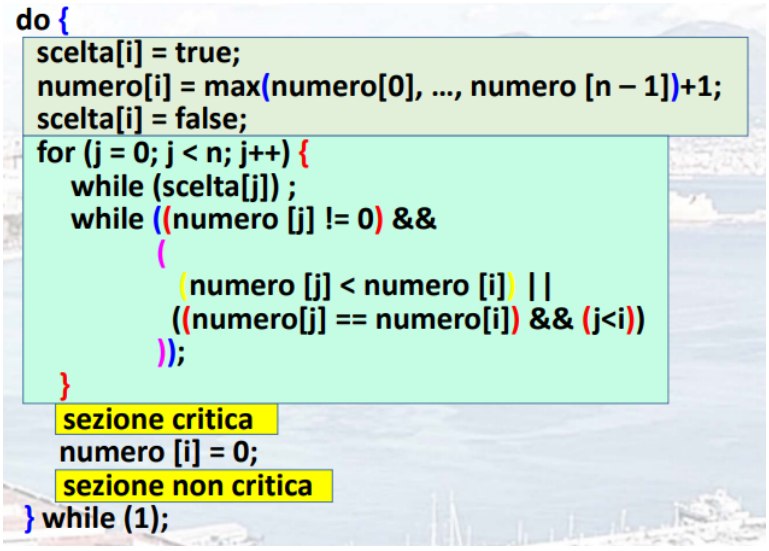
Se un processo ha già richiesto l'ingresso esiste un numero limite al numero di volte che si consente ad altri processi di entrare nelle rispettive sezioni critiche prima che si accordi la richiesta del prossimo processo.

Algoritmo del Fornaio

Prima di entrare nella sua sezione critiva il processo riceve un numero. Chi detiene il numero più basso entra nella sezione critica.

Se i due processi P_i e P_j ricevono lo stesso numero, se $i < j$ allora è servito prima P_i, altrimenti viceversa.

Si effettua quindi un controllo lessico-grafico utilizzando il PID



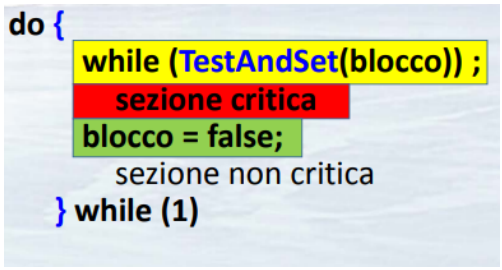
Architetture di sincronizzazione

Controllo e modifica del contenuto di una parola di memoria in modo atomico.

Mutua esclusione con TestAndSet

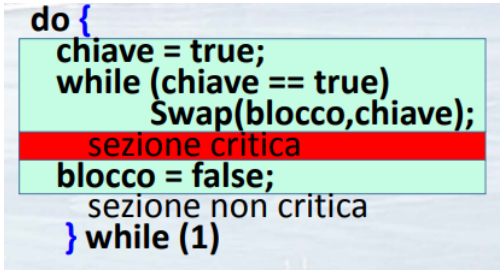
```
boolean TestAndSet(boolean &obiettivo){
  boolean valore = obiettivo;
```

```
  obiettivo = true;
  return valore;
}
```



Mutua esclusione con Swap

```
void Swap(boolean &a, boolean &b){
  boolean temp = a;
  a = b;
  b = temp;
}
```



Semafori

Un semaforo è uno strumento di sincronizzzazione che non richiede attesa attiva.

Un semaforo S è definita come una variabile intera, alla quale si può accedere solo tramite le due operazioni atomiche predefinite di wait e signal.

- wait → Comando di attesa e di verifica → Se la variabile $S \leq 0$ esegue un'operazione nulla.
- signal → Aumenta di 1 la variabile S

Sezione critica di n processi

Dati condivisi → semaphore mutex;

Processo P_i:

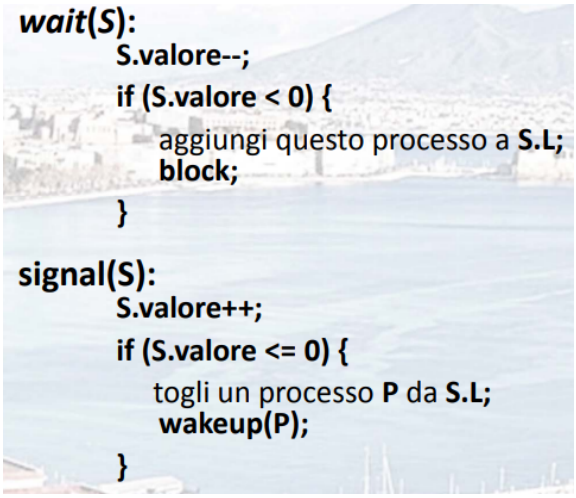
```
mutex = 1; // valore iniziale del semaforo
do{
  wait(mutex);
    // Sezione critica
  signal(mutex);
    // Sezione non critica
} while(1);
```

Realizzazione di semafori

```
typedef struct{
  int valore;
  struct processo *L;
} semaforo;
```

Si utilizzano due operazioni:

- block → sospende il processo che la invoca
- wakeup → riprende l'esecuzione di un processo bloccato



Semaforo come strumento generale di sincronizzazione

Si può usare un semaforo per stabilire la priorità di uno o più processi → Presi due processi P_i e P_j e utilizzando una flag inizializzata a 0:

Utilizzo signal e wait per stabilire quale operazione eseguire prima → Viene eseguita l'operazione B in P_j solo dopo che A è stato eseguito in P_i

Stallo e attesa indefinita

> Uno stallo, detto anche deadlock, è situazione in cui due o più processi attendono indefinitamente un evento che può essere

> causato solo da uno dei processi in attesa.

Si entra in uno stato di attesa indefinita, o starvation.

Tipi di semafori

- Semaforo contatore → Il suo valore intero può variare in un dominio logicamente non limitato.
- Semaforo binario → Il suo valore intero può essere soltanto 0 o 1.
  - Si può implementare un semaforo contatore come semaforo binario.

Problemi tipici di sincronizzazione

▼ Problema dei produttori e dei consumatori

Nel codice → piene = 0; vuote = n; mutex = 1;

```
do {
  wait(piene)
  wait(mutex);
  ...
  rimuovi un elemento da vettore e
  inseriscilo in da_consumare
  ...
  signal(mutex);
  signal(vuote);
  ...
  consuma l'elemento contenuto in
  da_consumare
  ...
} while (1);
```

Processo Produttore

▼ Problema dei lettori e degli scrittori

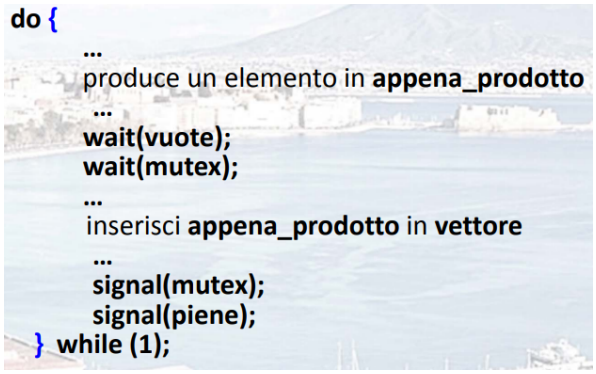
Inizialmente nel codice → mutex = 1; scrittura = 1; num_lettori = 0;

```
wait(scrittura);
...
esegui l'operazione di scrittura
...
signal(scrittura);
```

Processo Scrittore

```
wait(mutex);
numlettori++;
if (numlettori == 1)
   wait(scrittura);
signal(mutex);
    ...
  esegui l'operazione di lettura
    ...
wait(mutex);
numlettori – –;
if (numlettori == 0)
  signal(scrittura);
signal(mutex):
```

Processo Lettore



Processo Consumatore

▼ Problema dei cinque filosofi

Vedi dalle slide che è tutto un casino