Sincronizzazione

Le sezioni scritiche

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Parallelismo e sincronizzazione

- Ambiente di sviluppo
  - Programmazione **parallela** (tramite P o T)
  - Entità molto spesso **cooperanti**

Processo o Thread

- Problematiche
  - Necessità di manipolare dati condivisi
  - Si possono verificare **corse critiche**
  - Possono esistere tratti di codice **non rientranti**

Risultato dipendende dall'ordine di esecuzione

- Strategia risolutiva
  - **Sincronizzare** opportunamente P e T

Codice non interrompibile

Rendere i programmi indipendenti dalla velocità relativa di P e T

"To much milk problem"

Orario	Persona A	Persona B
10.00	Frigo? Finito latte.	
10.05	Va al negozio.	
10.10	Arriva al negozio.	Frigo? Finito latte.
10.15	Acquista il latte.	Va al negozio.
10.20	Arriva a casa.	Arriva al negozio.
10.25	Ritira il latte in frigo.	Acquista il latte.
10.30		Arriva a casa.
10.35		Ritira il latte in frigo.

LIFO - Stack

P_i / T_i

```
void push (int val) {
  if(top>=SIZE)
    return;
  stack[top] = val;
  top++;
  return;
}
```

P_j / T_j

```
void pop (int *val) {
  if(top<=0)
    return;
  top--;
  *val=stack[top];
  return;
}
```

- Le funzioni **push** e **pop**
  - Agiscono sulla stessa estremità dello stack
  - La variabile **top** è condivisa

top++ poi top-- o viceversa
Problemi?!

Possibile sovrascrivere o perdere una push, fare una pop di valore inesistente, etc.

FIFO – Queue – Buffer Circolare

P_i / T_i

```
void enqueue (int val) {
  if (n>SIZE) return;
  queue[tail] = val;
  tail=(tail+1)%SIZE;
  n++;
  return;
}
```

registro = n
registro = registro+1
n = registro

P_j / T_j

```
int dequeue (int *val) {
  if (n<=0) return;
  *val=queue[head];
  head=(head+1)%SIZE;
  n--;
  return;
}
```

registro = n
registro = registro-1
n = registro

- Le funzioni **enqueue** e **dequeue**
  - Agiscono su estremità "diverse" della coda usando variabili diverse **tail** e **head**
  - La variabile **n** è comunque condivisa

Possibile perdere un incremento o un decremento

Le sezioni critiche

- Sezione critica (**SC**) o regione critica (**RC**)
  - Una sezione di codice, comune a più P (o T), nella quale i P (o T) possono accedere (in lettura e **scrittura**) a oggetti comuni
- Ovvero una **SC** o **RC** è
  - Una sezione di codice nella quale più P (o T) competono per l'uso (in lettura e **scrittura**) di risorse comuni (e.g., dati condivisi)

Le sezioni critiche

- Le corse critiche potrebbero essere evitate se
  - Non si avessero mai più P (o T) nella stessa SC contemporaneamente
  - Quando un P (o T) è in esecuzione nella sua SC nessun altro P (o T) potesse fare altrettanto
  - Il codice nella SC fosse eseguito da un singolo P (o T) alla volta
  - L'esecuzione del codice nella SC fosse effettuato in **mutua esclusione**

Protocollo di accesso

- Soluzione
  - Per ciascuna SC, occorre stabilire un **protocollo** di accesso per forzare la **mutua esclusione**
- Ovvero
  - Per entrare in una SC un processo esegue codice di **prenotazione**
    - La prenotazione deve essere bloccante se la SC è utilizzata da un altro processo
  - Per uscire da una SC, un processo esegue codice di **rilascio** della regione occupata
    - Il rilascio sblocca altri P (o T) eventualmente in attesa

Protocollo di accesso

P_i / T_i

```
while (TRUE) {
  ...
  sezione d'ingresso
  SC
  sezione d'uscita
  ...
  sezione non critica
}
```

P_j / T_j

```
while (TRUE) {
  ...
  sezione d'ingresso
  SC
  sezione d'uscita
  ...
  sezione non critica
}
```

- Ogni SC è protetta da
  - Una sezione di ingresso (di prenotazione o prologo)
  - Una sezione di uscita (di rilascio)
- Sezioni critiche indipendenti devono essere protette separatamente
- Sezioni **non** critiche **non** devono essere protette

Condizioni

- Ogni soluzione al problema delle SC **deve** soddisfare i seguenti requisiti
  - Mutua esclusione (ME)
    - Un solo P (o T) alla volta deve ottenere l'accesso alla SC
  - Progresso
    - Se nessun P (o T) si trova nella SC e un P (o T) desidera entrarci, deve poterlo fare in un tempo definito
      - Solo i P (o T) in fase di prenotazione possono partecipare alla selezione
      - Nessun P (o T) fuori dalla SC può bloccare altri P (o T)
    - Ovvero occorre evitare **deadlock** tra P (o T)

Condizioni

- Attesa definita
  - Deve esistere un numero definito di volte per cui altri P (o T) riescano ad accedere alla SC prima che un P (o T) specifico e che ha fatto una richiesta di accesso possa farlo
  - Ovvero, occorre evitare **starvation** di P (o T)
- Ogni soluzione dovrebbe essere simmetrica
  - La selezione di chi deve accedere alla SC non dovrebbe dipendere dalla
    - Priorità relativa tra P (o T)
    - Velocità relativa dei P (o T)

Soluzioni

- Le SC ammettono soluzioni
  - Software
    - La correttezza risiede nella logica dell'**algoritmo** così come formulato dal programmatore
  - Hardware
    - La soluzione si basa su soluzioni **architetturali** particolari (e.g., istruzioni macchina atomiche)
  - Ad-Hoc
    - Il **sistema operativo** fornisce funzioni e strutture dati e il programmatore le utilizza in maniera opportuna

Semaforo: Concetto introdotto da Dijkstra [1965]