Architettura degli Elaboratori 2

Esercitazione 2

- Scheduling: grafo di allocazione delle risorse
- Synchronization: comunicazione tra processi

A. Memo - 2005

Concetti preliminari (1)

L'esecuzione dei programmi può essere:
- **SEQUENZIALE**, esecuzione sequenziale di istruzioni appartenenti ad un programma, processo unico
- **CONCORRENTE**, esecuzione parallela di due o più processi (o segmenti dello stesso processo)

L'ambiente di esecuzione può essere:
- **MULTIPROGRAMMING** (un processore, più processi)
- **MULTIPROCESSING** (più processori a memoria condivisa, più processi)
- **DISTRIBUTED PROCESSING** (più processori a memoria non condivisa, più processi)

Concetti preliminari (2)

l'avanzamento concorrente di più processi è regolamentato:
- dal Sistema Operativo, che decide chi e per quanto tempo deve avanzare o porsi in attesa (**scheduling**); di norma i processi non ne sono consapevoli
- dai programmi che determinano i processi, che decidono come e quando avanzare in base all'interazione con altri processi a loro noti (**synchronization**);

Concetti preliminari (3)

Le politiche di scheduling vengono utilizzate:
- per determinare il prossimo processo da attivare
- per regolamentare l'accesso alle risorse *(anche se ...)*

Problematiche dello scheduling:
- deadlock (stallo o blocco critico)
  - *blocco indefinito e reciproco di due o più processi*
- starvation (inedia o blocco indefinito)
  - *attesa indefinita di uno o più processi mentre gli altri avanzano*
- lockout

Grafo di allocazione delle risorse

- è uno strumento grafico utile per l'individuazione delle situazioni di stallo
- è caratterizzato da tre insiemi:
  - insieme dei processi del sistema $P = \{P_1,...,P_N\}$
  - insieme delle risorse del sistema $R = \{R_1^h,...,R_M^k\}$
  - insieme delle associazioni processo-risorsa $E = \{E_1,...,E_L\}$

Grafo di allocazione delle risorse

- un processo si rappresenta con un cerchio
- una risorsa si rappresenta con un rettangolo al cui interno sono presenti tanti punti di ancoraggio quanto è la molteplicità di istanza della risorsa
- un arco:
  - che collega il (bordo del) processo P_i al (punto di ancoraggio) della risorsa R_j, indicato con $e_R : P_i \rightarrow R_j$
  - che collega il punto di ancoraggio della risorsa R_j al bordo del processo P_i, indicato con $e_A : R_j \rightarrow P_i$

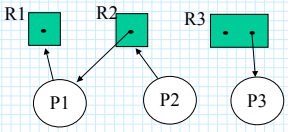
Grafo di allocazione delle risorse

esempio

$P = \{P_1, P_2, P_3\}$

$R = \{R_1^1, R_2^1, R_3^2\}$

$E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$



Grafo di allocazione delle risorse

se non ci sono percorsi chiusi (cicli):

- **non c'è sicuramente stallo**

se ci sono percorsi chiusi:

- se ci sono solo risorse unarie, allora **c'è sicuramente stallo**
- se ci sono anche risorse con molteplicità maggiore di 1, allora **va verificato di volta in volta**

Esercizio con soluzione

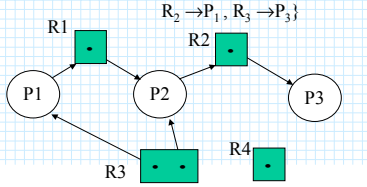
Dati:

$P = \{P_1, P_2, P_3\}$

$R = \{R_1^1, R_2^2, R_3^2, R_4^1\}$

$E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2$

$R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$



Grafo di allocazione delle risorse

Possibili esercizi:

- data la rappresentazione insiemistica
  - passare alla rappresentazione grafica
  - individuare lo stato attuale
  - valutare che cosa succede in particolari situazioni
- data la rappresentazione grafica
  - passare alla rappresentazione insiemistica
  - individuare lo stato attuale
  - valutare che cosa succede in particolari situazioni

Esercizio 1

Dato il sistema descritto dalla seguente rappresentazione insiemistica delle assegnazione delle risorse

$P = \{P_1, P_2, P_3, P_4, P_5\}$

$R = \{R_1^2, R_2^2, R_3^1, R_4^1\}$

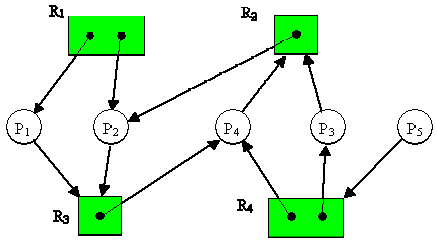
$E = \{P_1 \rightarrow R_3, P_2 \rightarrow R_3, P_3 \rightarrow R_2, P_4 \rightarrow R_2, P_5 \rightarrow R_4,$

$R_1 \rightarrow P_1, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_3 \rightarrow P_4, R_4 \rightarrow P_3, R_4 \rightarrow P_4\}$

rappresentare il relativo grafo di allocazione delle risorse, e verificare se il sistema si trova in condizione di stallo. Successivamente verificare se la situazione cambia con il rilascio da parte del processo P_3 della risorsa R_4.

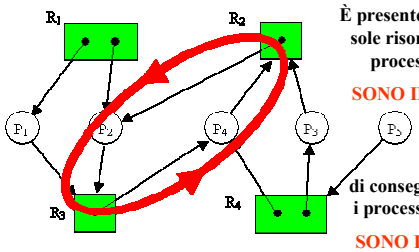
Esercizio 1 (soluzione)

1. rappresentazione del relativo grafo di allocazione delle risorse



Esercizio 1 (soluzione)

2. individuazione di eventuali cicli



È presente un ciclo con sole risorse unarie: i processi P_2 e P_4

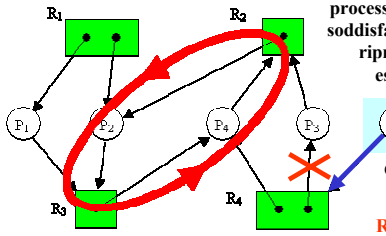
SONO IN STALLO

di conseguenza anche i processi P_1 , P_3 e P_5

SONO IN STALLO

Esercizio 1 (soluzione)

3. analisi di una variazione



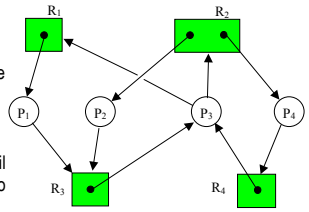
Ora la richiesta del processo P_5 può essere soddisfatta, e quindi P_5 riprende la sua esecuzione.

Gli altri processi P_1 , P_2 , P_3 e P_4

RIMANGONO IN STALLO

Esercizio 2

Dato il sistema descritto dal seguente grafo di allocazione delle risorse dedurre la relativa rappresentazione insiemistica delle assegnazione delle risorse, e verificare se il sistema si trova in condizione di stallo. Successivamente verificare se la situazione cambia con il rilascio da parte del processo P_2 della risorsa R_2.



Esercizio 2

1. rappresentazione dei relativi insiemi di assegnazione delle risorse

$P = \{P_1, P_2, P_3, P_4\}$

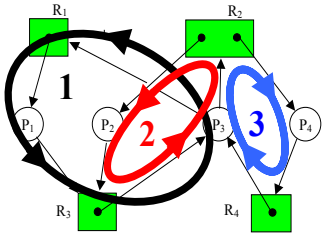
$R = \{R_1^1, R_2^2, R_3^1, R_4^1\}$

$E = \{P_1 \rightarrow R_3, P_2 \rightarrow R_3, P_3 \rightarrow R_1, P_3 \rightarrow R_2, P_4 \rightarrow R_4,$

$R_1 \rightarrow P_1 , R_2 \rightarrow P_2, R_2 \rightarrow P_4, R_3 \rightarrow P_3 , R_4 \rightarrow P_3\}$

Esercizio 2

2. individuazione di eventuali cicli



CICLO 1

$P_1 \rightarrow R_3 \rightarrow P_3 \rightarrow R_1 \rightarrow P_1$

STALLO SICURO

CICLO 2

$P_3 \rightarrow R_2 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3$

DA VERIFICARE

STALLO

CICLO 3

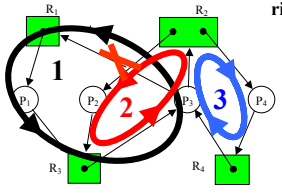
$P_3 \rightarrow R_2 \rightarrow P_4 \rightarrow R_4 \rightarrow P_3$

DA VERIFICARE

STALLO

Esercizio 2

3. analisi di una variazione

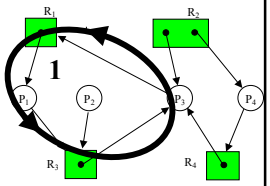


il CICLO 1 persiste, quindi

$P_1 \rightarrow R_3 \rightarrow P_3 \rightarrow R_1 \rightarrow P_1$

STALLO SICURO

La risorsa R_2 può soddisfare la richiesta di P_3 , si eliminano due cicli e si ottiene



e i processi P_2 e P_4 ?

Sincronizzazione

Per poter interagire, i processi devono comunicare

La comunicazione serve per
- *scambiare dati*
- *sincronizzare l'avanzamento*

La comunicazione avviene tramite:
- *variabili lucchetto*
- *semafori*
- *altro ...*
- *monitor*
- *messaggi*

Comunicazione tra processi

- **race conditions**, situazioni in cui l'accesso simultaneo ad una variabile condivisa determina la non prevedibilità del suo valore
- individuazione delle **sezioni critiche**, nelle quali i processi accedono a potenziali *race conditions*
- soluzione: **mutua esclusione** nell'accesso alle sezioni critiche

Comunicazione tra processi

tecniche per ottenere la **mutua esclusione**:
- disabilitare gli interrupt
- uso di un'istruzione atomica (TSL)
- variabile lucchetto (da sola non è sufficiente)
- alternanza stretta (solo se i due processi si alternano rigidamente)

se è garantita l'atomicità funzionano, ma inducono attesa attiva (uso di *sleep()* e *wakeup()*)

Comunicazione tra processi

Semaforo
- è una variabile intera S, a cui è possibile accedere solo tramite funzioni atomiche wait() e signal() [originariamente P() da *proberen* -testare- e V() da *verhogen* –incrementare]

wait(S)	while (S<=0) sleep(); S--;	*se **S=0** risorsa occupata* *se **S=1** risorsa libera*
signal(S)	S++; wakeup();	*se **S=-1** risorsa occupata e un processo in coda*

Comunicazione tra processi

Semaforo binario o mutex
- è un semaforo in cui S può valere solo 0 o 1
- viene utilizzato per garantire la mutua esclusione
- con un semaforo binario è possibile implementare anche semafori a contatore

esempio di implementazione di un semaforo binario:

segue %

```
Class SemaforoBinario {
    private int valore;
    Queue coda0 = new Queue();
    Queue coda1 = new Queue();
    SemaforoBinario() {
        valore = 1;
    }
    void wait() {
        if (valore == 0) {
            coda0.add(<processo>);
            suspend(<processo>);
        }
        valore --;
        if (coda1.size() > 0)
            wakeup(coda1.remove());
    }
    void signal() {
        if (valore == 1) {
            coda1.add(<processo>);
            suspend(<processo>);
        }
        valore ++;
        if (coda0.size() > 0)
            wakeup(coda0.remove());
    }
}
```

Mutua esclusione con semafori

```
semaforo mutex = 1; // semaforo libero
void Processo_N() {
  while (true) {
     zona_non_critica();
     wait(mutex);
     zona_critica();
     signal(mutex);
     zona_non_critica();
  }
}
```

Lettori - scrittori

- un grande Data Base condiviso da molti utenti
- qualcuno vuole solo leggerne i valori (lettori)
- qualcuno vuole leggere e/o scrivere valori (scrittori)
- è ammessa la lettura multipla, ma per la scrittura può accedervi un solo scrittore per volta, e gli altri lettori vengono esclusi

è il modello dell'accesso ad un DataBase

```
semaforo mutex = 1;
semaforo db = 1;
int lettori = 0;
void scrittore (void) {
  while (true) {
     pensa();
     P(db);
     scrivi();
     V(db);
  }
}
```

```
void lettore (void) {
  while (true) {
     P(mutex);
     lettori++;
     if (lettori==1) P(db);   *
     V(mutex);
     leggi();                 **
     P(mutex);
     lettori--;
     if (lettori==0) V(db);   *
     V(mutex);                **
     usa_i_dati_letti();
  }
}
```

* *il primo lettore blocca l'accesso di eventuali scrittori (occupa db)*

** *effettua la lettura in maniera non mutuamente esclusiva*

* *l'ultimo lettore sblocca l'accesso*

** *ad eventuali scrittori (libera db)*

*il semaforo **mutex** serve per assicurare la mutua esclusione nell'aggiornamento della variabile* lettori

*il **db** è il semaforo di mutua esclusione per gli scrittore (uno solo alla volta) e per i lettori (anche più di uno)*

la variabile **lettori** *indica il numero di processi che stanno leggendo i dati*

Produttore - consumatore

- un processo (produttore) genera dati che dovranno essere utilizzati da un secondo processo (consumatore)
- i dati emessi dal produttore vengono salvati in un buffer di dimensioni finite, da dove vengono prelevati dal consumatore
- se il buffer è pieno, il produttore deve bloccarsi in attesa che si liberi un posto
- se il buffer è vuoto, il consumatore deve bloccarsi in attesa che arrivi un nuovo dato

Produttore - consumatore

soluzione:

- la risorsa condivisa, a cui va garantito l'accesso mutuamente esclusivo è il buffer, e quindi le operazioni di inserisci() e preleva()
- al fine di regolamentare l'utilizzo del buffer, devono essere previsti due meccanismi che sospendano e facciano riprendere le attività dei due processi in concomitanza degli eventi buffer vuoto e buffer pieno

```
int N = … ;  numero di posti nel buffer
semaforo mutex = 1;  controlla l'accesso in sezione critica
semaforo vuoti = N;  numero di posti liberi disponibili nel buffer
semaforo pieni = 0;  numero di posti già occupati  nel buffer
```

```
void produttore (void) {
   int prod;
   while (true) {
      prod = produci();
      P(vuoti);
      P(mutex);
      inserisci(prod);
      V(mutex);
      V(pieni);
   }
}
```

```
void consumatore (void) {
   int prod;
   while (true) {
      P(pieni);
      P(mutex);
      prod = preleva();
      V(mutex);
      V(vuoti);
      consuna(prod);
   }
}
```

SOLUZIONE ERRATA con stallo

```
void produttore (void) {
   int prod;
   while (true) {
      prod = produci();
1     P(mutex);
      P(vuoti);
2     inserisci(prod);
      V(mutex);
      V(pieni);
   }
}
```

```
void consumatore (void) {
   int prod;
   while (true) {
3     P(pieni);
4     P(mutex);
      prod = preleva();
      V(mutex);
      V(vuoti);
      consuna(prod);
   }
}
```

1) *il produttore entra in sezione critica, e impedisce al consumatore di entrarvi*
2) *il produttore trova il buffer pieno (non ci sono slot liberi) e va in wait()*
3) *il consumatore vede che ci sono spazi pieni ed avanza*
4) *il consumatore si ferma perché la zona critica è già occupata*

SOLUZIONE ERRATA SENZA SEMAFORI

```
void produttore (void) {
   int prod;
   while (true) {
      prod = produci();
      if (count == N)
            sleep();
      inserisci(prod);
      count++;
      if (count == 1)
            wakeup();
   }
}
```

si può manifestare race condition

```
void consumatore (void) {
   int prod;
   while (true) {
      if (count == 0)
            sleep();
      prod = preleva();
      count--;
      if (count == N-1)
            wakeup(produttore);
      consuna(prod);
   }
}
```

Il Barbiere dormiente

In un negozio di barbiere c'è un unico barbiere, una sola poltrona da lavoro ed N sedie d'attesa per i clienti.

- se non ci sono clienti il barbiere dorme sulla poltrona
- quando arriva un cliente, sveglia il barbiere; se ne arrivano altri si accomodano sulle sedie d'attesa
- se si riempiono tutte le sedie, i nuovi arrivati se ne vanno

```
int attesa = 0;  numero clienti in attesa
semaforo mutex = -1;  mutua esclusione alla variabile 'attesa'
semaforo clienti = 0;  numero di clienti in attesa
semaforo barbiere = 0;  numero di attività del barbiere
```

```
void barbiere (void) {
   while (true) {
1     P(clienti);
      P(mutex);
      attesa--;
3     V(barbiere);
      V(mutex);
      taglio_capelli();
   }
}
```

```
void cliente (void) {
      P(mutex);
5     if ( attesa < N ) {
         attesa++;
         V(clienti);
         V(mutex);
         P(barbiere);
         subisce_taglio();
      } else  V(mutex);
   }
```

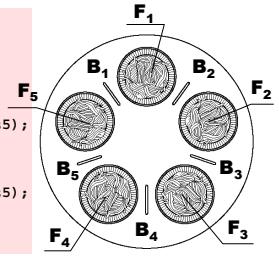
1) *si pone in attesa di clienti*
2) *accesso esclusivo ad attesa*
3) *Il barbiere inizia il taglio*
4) *rilascia l'accesso ad attesa*
5) *se ci sono posti liberi …*
6) *avvisa il barbiere che ci sono clienti*
7) *chiede al barbiere di iniziare*
8) *negozio pieno, non entra*

I Filosofi cinesi a pranzo

- 5 filosofi seduti attorno ad una tavola rotonda imbandita
- in tavola ci sono 5 piatti sempre pieni di spaghetti
- tra un piatto e l'altro c'è un bastoncino cinese
- i filosofi alternano continuamente 2 fasi di durata casuale: nella prima pensano e nella seconda mangiano
- per mangiare un filosofo abbisogna di entrambi i bastoncini ai suoi lati

I Filosofi cinesi a pranzo

```
void filosofo (int x) {
   while (true) {
      <pensa>;
      PrendiBacchetta(x);
      PrendiBacchetta((x+1)%5);
      <mangia>;
      LasciaBacchetta(x);
      LasciaBacchetta((x+1)%5);
   }
}
```



SOLUZIONE ERRATA

I Filosofi cinesi a pranzo

possibile soluzione:

- prima di prendere la seconda bacchetta, controllare se è disponibile
  - se si, prelevarla e mangiare
  - se no, non porsi in attesa e rilasciare anche la prima
- elimina il deadlock, ma non la starvation
- per ridurre la starvation si potrebbe introdurre un ritardo casuale prima di ritentare (Ehernet)

I Filosofi cinesi a pranzo

Soluzione corretta 1:

- dato che le bacchette sono condivise, la sezione critica, che comprende sia l'accesso che il rilascio delle bacchette, può essere regolamentata da un unico semaforo mutex tra tutti i filosofi
- scarsa efficienza dato che può mangiare un solo filosofo per volta

I Filosofi cinesi a pranzo

Soluzione corretta 2:

- ogni filosofo, prima di mangiare, entra in uno stato mutuamente esclusivo in cui controlla se i filosofi adiacenti stanno mangiando o meno:
  - se anche uno solo di loro sta mangiando, aspetta che finisca
  - se entrambi stanno pensando, acquisisce le bacchette e libera il mutex

```
int PENSA = 0;
int ASPETTA = 1;
int MANGIA = 2;
int
Semaforo mutex = new Semaforo(1);
Semaforo s[5] = new Semaforo[5];

void
   w
   }
}
void
   m
   s
   prova(SIN);
   prova(DES);
   s[x].down();
}
   stato[x] = MANGIA;
   s[x].up();
   }
}
```

mutex	semaforo binario di mutua esclusione per l'accesso alla sezione critica: solo un filosofo alla volta può entrare in zona critica e cercare di accedere alle bacchette
s[5]	vettore di semafori, uno per ogni filosofo
stato[5]	vettore di interi che contiene lo stato dei rispettivi filosofi; valori possibili: 0 (PENSA), 1 (ASPETTA), 2 (MANGIA)
PrendiBacchette()	tenta di prendere le bacchette: se sono occupate si pone in attesa
LasciaBacchette()	rilascia entrambe le bacchette

```
int PENSA = 0;
int ASPETTA = 1;
int MANGIA = 2;
int stato[] = new int[5];
Semaforo mutex = new Semaforo(1);
Semaforo s[5] = new Semaforo[5];

sostituire   con
SIN          (x+4)%5
DES          (x+1)%5

void filosofo (int x) {
   while (true) {
      <pensa>;
      PrendiBacchette(x);
      <mangia>;
      LasciaBacchette(x);
   }
}
void LasciaBacchette(int x) {
   mutex.down();
   stato[x] = PENSA;
   prova(SIN);
   prova(DES);
   s[x].down();
}

void PrendiBacchette(int x) {
   mutex.down();
   stato[x] = ASPETTA;
   prova(x);
   mutex.up();
   s[x].down();
}
void prova(int x) {
   if ( stato[x] == ASPETTA &&
        stato[SIN] != MANGIA &&
        stato[DES] != MANGIA)  {
      stato[x] = MANGIA;
      s[x].up();
   }
}
```