Architettura degli Elaboratori 2

Esercitazioni 2

- Grafo di allocazione delle risorse
- Comunicazione tra processi

A. Memo - 2004

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 $e_A : R_j \rightarrow P_i$

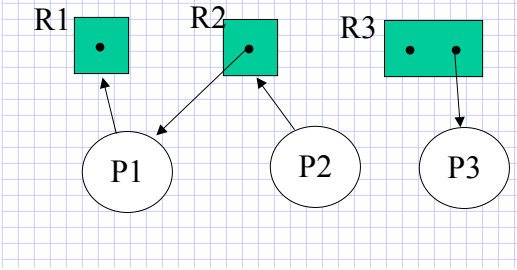
Grafo di allocazione delle risorse

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 (detti anche cicli), allora 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

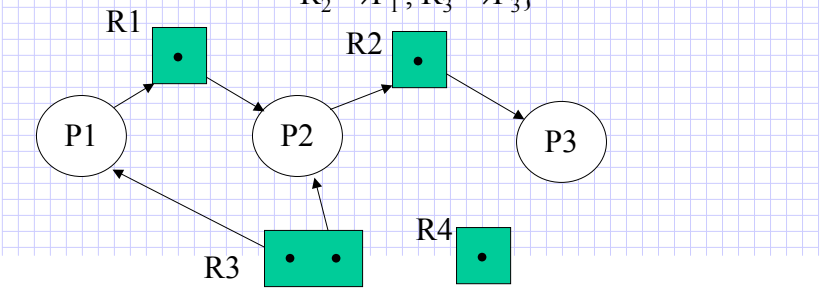
Dati:

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

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

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

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



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*

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 aspetta di poter scrivere ed occupa db*

** *effettua la lettura in maniera mutuamente esclusiva*

*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 e per il primo lettore nell'entrata e l'ultimo lettore nell'uscita dalla sezione critica*

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);
      P(mutex);
4     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

si può manifestare race condition

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

```
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 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 sulla sedia
semaforo clienti = 0;  numero di clienti in attesa
semaforo barb = 1;  stato di attività del barbiere
void barbiere (void) {
   while (true) {
      P(clienti);
      P(mutex);
      attesa--;
      V(barb);
      V(mutex);
      taglio_capelli();
   }
}
```

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

1) *si pone in attesa di clienti*
2) *accesso esclusivo ad attesa*
3) *in attesa del barbiere*
4) *rilascia l'accesso esclusivo ad attesa*
5) *se ci sono posti liberi …*
6) *sveglia il barbiere*