Architettura degli Elaboratori 2

Esercitazioni 1

- Scheduling della CPU
- Grafo di allocazione delle risorse
- Comunicazione tra processi

Scheduling della CPU

- consiste nel selezionare un processo dalla *ready list* e attribuirgli la CPU
- l'operazione viene effettuata dal dispatcher
- possibili algoritmi di scheduling:
  - First Come First Served [FCFS]
  - Round Robin [RR]
  - Shortest Job First [SJF]
  - con priorità esterna

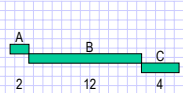
First Come First Served

- la CPU viene assegnata al processo che la richiede per primo (viene alimentata con una coda FIFO)

esempio — processo A: tempo di esecuzione = 2 [u.t.]
processo B: tempo di esecuzione = 12 [u.t.]
processo C: tempo di esecuzione = 4 [u.t.]

N.B. = trascuriamo per semplicità i tempi di scambio di contesto

First Come First Served



TEMPO DI ATTESA

T_{att} (A) = 0

T_{att} (B) = 2

T_{att} (C) = 2 + 12 = 14

T_{att} (medio) = (0 + 2 + 14) / 3 = 5,3 [u.t.]

TEMPO DI TURN AROUND

T_{ta} (A) = 2

T_{ta} (B) = 2 + 12 = 14

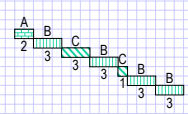
T_{ta} (C) = 2 + 12 + 4 = 18

T_{ta} (medio) = (2 + 14 + 18) / 3 = 11,3 [u.t.]

Round Robin

- come l'FCFS, ma con prelazione per esaurimento del quanto di tempo (la *ready list* viene trattata come una coda circolare)

Round Robin



time slice = 3 [u.t.]

TEMPO DI ATTESA

T_{att} (A) = 0

T_{att} (B) = 2 + 3 + 1 = 6

T_{att} (C) = 2 + 3 + 3 = 8

T_{att} (medio) = (0 + 6 + 8) / 3 = 4,6 [u.t.]

TEMPO DI TURN AROUND

T_{ta} (A) = 2

T_{ta} (B) = 2 + 3 + 3 + 3 + 1 + 3 +3 = 18

T_{ta} (C) = 2 + 3 + 3 + 3 + 1 = 12

T_{ta} (medio) = (2 + 18 + 12) / 3 = 10,6 [u.t.]

Round Robin

Provate a calcolare i tempi di attesa e di turnaround medi con un valore di time slice pari a 1e 5 [u.t.]. Cambierà qualcosa?

time slice = 1 [u.t.]

T_{att} (medio) = (2 + 6 + 6) / 3 = 4,6 [u.t.] T_{ta} (medio) = (4 + 18 + 10) / 3 = 10,6 [u.t.]

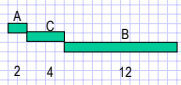
time slice = 5 [u.t.]

T_{att} (medio) = (0 + 2 + 7) / 3 = 3 [u.t.] T_{ta} (medio) = (2 + 18 + 11) / 3 = 10,3 [u.t.]

Shortest Job First

- o meglio *Shortest next-CPU-burst First*
- la CPU viene assegnata al processo che ha il *CPU-burst* successivo più breve
- può essere *non preemptive*
- o *preemptive* (SRTF)

Shortest Job First



TEMPO DI ATTESA

T_{att} (A) = 0

T_{att} (B) = 2 + 4 = 6

T_{att} (C) = 2

T_{att} (medio) = (0 + 6 + 2) / 3 = 2,6 [u.t.]

TEMPO DI TURN AROUND

T_{ta} (A) = 2

T_{ta} (B) = 2 + 4 + 12 = 18

T_{ta} (C) = 2 + 4 = 6

T_{ta} (medio) = (2 + 18 + 6) / 3 = 8,6 [u.t.]

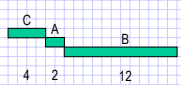
con Priorità (esterna)

- ad ogni processo viene attribuita una priorità e la CPU viene assegnata al processo con priorità più elevata

esempio processo A: priorità = 2
processo B: priorità = 1
processo C: priorità = 3

N.B. = 3 è il valore di priorità più elevata

con Priorità (esterna)



TEMPO DI ATTESA

T_{att} (A) = 4

T_{att} (B) = 4 + 2 = 6

T_{att} (C) = 0

T_{att} (medio) = (4 + 6 + 0) / 3 = 3,1 [u.t.]

TEMPO DI TURN AROUND

T_{ta} (A) = 4 + 2 = 6

T_{ta} (B) = 4 + 2 + 12 = 18

T_{ta} (C) = 4

T_{ta} (medio) = (6 + 18 + 4) / 3 = 9,3 [u.t.]

Esercizio con soluzioni

Cinque processi batch, idendificati dalle lettere da A a E, arrivano al calcolatore approssimativamente allo stesso istante. I processi hanno un tempo di esecuzione stimato di 8, 10, 2, 4 e 8 unità di tempo rispettivamente, mentre le loro priorità (determinate esternamente) sono rispettivamente 2, 4, 5, 1 e 3 (dove 5 è la priorità massima). Per ognuno dei seguenti algoritmi di schedulazione determinare il *tempo medio di turnaround* e il *tempo medio di attesa*, trascurando i tempi dovuti allo scambio di contesto.

- Round Robin (time slice = 2) in un sistema multiprogrammato
- con priorità esterna (un processo per volta, fino al completamento)
- FCFS (un processo per volta, fino al completamento)
- SJF (un processo per volta, fino al completamento)

soluzioni

RR

t_{att} (medio) = 15,6 [u.t.]
t_{ta}(medio) = 22 [u.t.]

priorità

t_{att} (medio) = 12,4 [u.t.]
t_{ta}(medio) = 18,8 [u.t.]

FCFS

t_{att} (medio) = 14 [u.t.]
t_{ta}(medio) = 20,4 [u.t.]

SJF

t_{att} (medio) = 8,8 [u.t.]
t_{ta}(medio) = 15,2 [u.t.]

Esercizio

Si supponga che tre clienti arrivino ad una stazione di servizio per fare il pieno di benzina, e che ognuna impieghi il seguente tempo (noto a priori)

auto	arrivo	servizio (in minuti)
A	8:00	8
B	8:06	5
C	8:07	2

Nell'ipotesi che alle 8:00 l'unica pompa di benzina della stazione sia libera, calcolare il tempo medio di attesa ed il tempo medio di turnaround nel caso di politiche di schedulazione FIFO, SJF non preemptive ed SJF preemptive.

Esercizio

Cinque processi batch, idendificati dalle lettere da A a E, arrivano al calcolatore approssimativamente allo stesso istante. I processi hanno un tempo di esecuzione stimato di 10, 6, 2, 4 e 8 unità di tempo rispettivamente, mentre le loro priorità (determinate esternamente) sono rispettivamente 3, 5, 2, 1 e 4 (dove 5 è la priorità massima). Per ognuno dei seguenti algoritmi di schedulazione determinare il *tempo medio di turnaround* e il *tempo medio di attesa*, trascurando i tempi dovuti allo scambio di contesto.

- Round Robin (*time slice* = 2) in un sistema multiprogrammato
- con priorità esterna (un processo per volta, fino al completamento)
- FCFS (un processo per volta, fino al completamento)
- SJF (un processo per volta, fino al completamento)

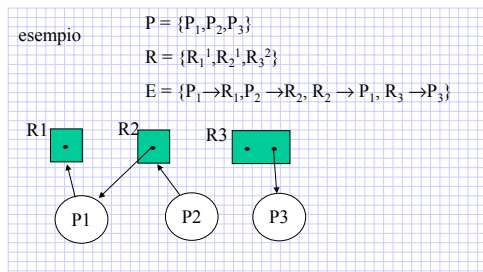
Grafo di allocazione delle risorse

- 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



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

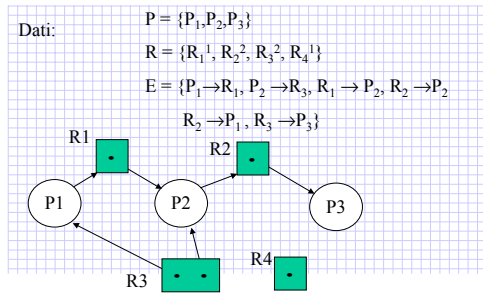
Esercizio con soluzione

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);
(2)   P(vuoti);
      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

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*