Mutua esclusione

Arturo Carpi

Dipartimento di Matematica e Informatica
Università di Perugia

Corso di Sistemi Operativi - a.a. 2021/22

Gestione dei processi

Multiprogrammazione	molti processi, un singolo processore
Multiprocessing	molti processi, un multiprocessore
Processi distribuiti	molti processi, molti processori distribuiti (cluster)

La concorrenza appare in tre differenti contesti:

- Applicazioni multiple
- Applicazioni strutturate
- Struttura del sistema operativo

Principi della concorrenza

Elaborazione concorrente

- Nell'uniprocessore multiprogrammato, le esecuzioni dei processi si alternano
- nel multiprocessore, le esecuzioni dei processi si alternano e si sovrappongono
- in entrambi i casi, la velocità relativa di esecuzione dei processi non è prevedibile. Dipende da:
  - attività degli altri processi
  - gestione delle interruzioni
  - politiche di schedulazione

Difficoltà

- condivisione di risorse globali
- allocazione ottimale delle risorse
- debugging

Esempio

```
void echo()
{
   chin = getchar();
   chout = chin;
   putchar(chout);
}
```

Nell'uniprocessore:

```
Processo P1
     •
chin = getchar();
  <prerilascio>
     •
     •
     •
   <esecuzione>
chout = chin;
putchar(chout);
     •
```

```
 Processo P2
     •
     •
  <esecuzione>
chin = getchar();
chout = chin;
putchar(chout);
  <prerilascio>
     •
     •
     •
```

```
Esito:
stampa due volte
il secondo
carattere letto
```

Esempio

Se un solo processo alla volta potesse eseguire la procedura:

```
Processo P1
     •
chin = getchar();
  <prerilascio>
     •
     •
  <esecuzione>
chout = chin;
putchar(chout);
     •
  <sospensione>
     •
     •
     •
     •
```

```
Processo P2
     •
     •
  <esecuzione>
  <sospensione>
     •
     •
     •
  <attivazione>
     •
  <esecuzione>
chin = getchar();
chout = chin;
putchar(chout);
     •
```

```
Esito:
stampa prima il
carattere richiesto
da P1, poi quello
richiesto da P2
```

Esempio

```
void echo()
{
   chin = getchar();
   chout = chin;
   putchar(chout);
}
```

Nel multiprocessore:

```
Processo P1
     •
chin = getchar();
     •
chout = chin;
putchar(chout);
     •
     •
```

```
Processo P2
     •
     •
chin = getchar();
chout = chin;
     •
putchar(chout);
     •
```

```
Esito:
stampa due volte
il secondo
carattere letto
```

Conclusione

- È necessario proteggere le variabili condivise
- Si deve controllare il codice che accede alla variabile

Race condition

Si verifica quando più processi o threads leggono e scrivono dati in modo che il risultato finale dipende dall'ordine di esecuzione delle istruzioni dei processi

Esempi

I processi P1 e P2 assegnano rispettivamente 1 e 2 alla variabile condivisa a. Il valore finale sarà quello del 'secondo arrivato'

I processi P1 e P2 condividono le variabili b e c con valori iniziali $b = 1$, $c = 2$. P1 esegue $b = b + c$, P2 esegue $c = b + c$.
Se arriva primo P1, avremo $b = 3,\ c = 5$. Nel caso opposto, $b = 4,\ c = 3$.

Problemi determinati dalla concorrenza

Sono compiti del sistema operativo:

- Tenere traccia dei processi
  - PCB
- Allocare e deallocare risorse per i processi attivi
  - Tempo di elaborazione
  - memoria
  - files
  - dispositivi I/O
- Proteggere i dati e le risorse di ogni processo da interferenze involontarie di altri processi
- Il risultato di ogni processo deve essere indipendente dalla sua velocità relativa a quella dei processi concorrenti

Degree of Awareness	Relationship	Influence that One Process Has on the Other	Potential Control Problems
Processes unaware of each other	Competition	• Results of one process independent of the action of others • Timing of process may be affected	• Mutual exclusion • Deadlock (renewable resource) • Starvation
Processes indirectly aware of each other (e.g., shared object)	Cooperation by sharing	• Results of one process may depend on information obtained from others • Timing of process may be affected	• Mutual exclusion • Deadlock (renewable resource) • Starvation • Data coherence
Processes directly aware of each other (have communication primitives available to them)	Cooperation by communication	• Results of one process may depend on information obtained from others • Timing of process may be affected	• Deadlock (consumable resource) • Starvation

Competizione per le risorse

Processi concorrenti sono in conflitto per l'uso delle risorse

Problemi

- Mutua esclusione
  - risorsa critica
  - sezione critica
- Stallo (deadlock)
- Starvation

Mutua esclusione

```
     /* PROCESS 1 */
void P1
{
  while (true) {
    /* preceding code */;
    entercritical (Ra);
    /* critical section */;
    exitcritical (Ra);
    /* following code */;
  }
}
```

```
     /* PROCESS 2 */
void P2
{
  while (true) {
    /* preceding code */;
    entercritical (Ra);
    /* critical section */;
    exitcritical (Ra);
    /* following code */;
  }
}
```

Cooperazione per condivisione

- Processi che interagiscono senza conoscersi
- Usano e modificano dati condivisi
- Devono cooperare per la corretta gestione dei dati condivisi
- Il meccanismo di controllo deve assicurare l'integrità dei dati
- Si presentano i problemi di mutua exclusione, stallo, starvation
  - Ma le operazioni di lettura non richiedono mutua esclusione

Esempio

```
P1:
     a = a + 1;          a = a + 1;
     b = b + 1;          b = 2 * b;
P2:                      b = b + 1;
     b = 2 * b;          a = 2 * a;
     a = 2 * a;
```

Cooperazione per scambio di messaggi

- I processi collaborano per un obiettivo comune
- La comunicazione permette di sincronizzarsi e coordinarsi
- Primitive per invio e ricezione di messaggi sono fornite dal kernel o dal linguaggio di programmazione
- Non richiede mutua esclusione
- Può determinare stallo o starvation.

Requisiti per la mutua esclusione

- Un solo processo alla volta è ammesso alla sezione critica per ciascuna risorsa condivisa.
- Un processo che si ferma fuori dalla sua sezione critica non deve interferire con gli altri processi
- Si deve evitare deadlock e starvation
- Se nessun processo è nella sezione critica, deve essere concesso a ogni processo di entrare senza ritardo
- Nessuna ipotesi sulla velocità relativa dei processi e sul numero dei processori
- Un processo resta nella sezione critica per un tempo finito

Mutua esclusione con supporto hardware

Disabilitazione delle interruzioni

- Su uniprocessore, assicura che la sezione critica sia portata a termine senza interruzioni:

```
while (true) {
    /* disabilita interruzioni */;
    /* sezione critica */;
    /* abilita interruzioni */;
    /* resto del programma */;
}
```

- primitive definite dal nucleo
- rallenta il sistema
- inapplicabile con multiprocessore

Mutua esclusione con supporto hardware - 2

Istruzioni speciali

- Un solo processore alla volta può accedere a una locazione di memoria
- speciali istruzioni macchina eseguono due operazioni su un dato in maniera atomica

Compare and swap

Equivalente a

```
int compare_and_swap ( int *word, int testval, int newval)
{
    int oldval;
    oldval = *word
    if (oldval == testval) *word = newval;
    return oldval;
}
```

```
/* program mutualexclusion */
const int n = /* number of processes */;
int bolt;
void P( int i)
{
   while (true) {
      while (compare_and_swap(bolt, 0, 1) == 1)
         /* do nothing */;
      /* critical section */;
      bolt = 0;
      /* remainder */;
   }
}

void main()
{
   bolt = 0;
   parbegin (P(1), P(2), ... ,P(n));
}
```

```
/* program mutualexclusion */
int const n = /* number of processes */;
int bolt;
void P( int i)
{
   int keyi = 1;
   while (true) {
      do exchange (&keyi, &bolt)
      while (keyi != 0);
      /* critical section */;
      bolt = 0;
      /* remainder */;
   }
}

void main()
{
  bolt = 0;
  parbegin (P(1), P(2), ..., P(n));
}
```

Istruzioni macchina speciali

Vantaggi

- applicabile a qualsiasi numero di processi su uniprocessore o multiprocessore
- semplicità
- può supportare multiple sezioni critiche

Svantaggi

- attesa attiva (busy-waiting)
- possibilità di stallo
- possibilità di starvation