Sincronizzazione

Soluzioni software

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Specifiche

- Le soluzioni software al problema della SC si basano sull'utilizzo di **variabili globali**
  - Disponibili nei sistemi a memoria condivisa
- Analizzeremo il caso con due soli P (o T)
  - Denominati P_i (T_i) e P_j (T_j)
    - Dato **i** allora **j=1-i** e viceversa
    - Ovvero se **i=0** allora **j=1-i=1** e viceversa
  - Casi con più di due P (o T) sono complessi in quanto le procedure analizzate non sono facilmente generalizzabili

Inoltre supporremo esistano i valori logici TRUE (1) e FALSE (0)

Mutua esclusione: Soluzione 1

- Variabili globali
  - int flag[2] = {FALSE, FALSE};

P_i / T_i

```
while (TRUE) {
   while (flag[j]);
   flag[i] = TRUE;
   SC
   flag[i] = FALSE;
   sezione non critica
}
```

P_j / T_j

```
while (TRUE) {
   while (flag[i]);
   flag[j] = TRUE;
   SC
   flag[j] = FALSE;
   sezione non critica
}
```

Mutua esclusione
Progresso
Attesa definita
Simmetria

?

Mutua esclusione: Soluzione 1

- Variabili globali
  - int flag[2] = {FALSE, FALSE};

P_i / T_i

```
while (TRUE) {
  while (flag[j]);
  flag[i] = TRUE;
  SC
  flag[i] = FALSE;
  sezione non critica
}
```

P_j / T_j

```
while (TRUE) {
  while (flag[i]);
  flag[j] = TRUE;
  SC
  flag[j] = FALSE;
  sezione non critica
}
```

- Mutua esclusione **non** assicurata
  - P_i e P_j possono accedere alla SC contemporaneamente

Mutua esclusione: Soluzione 1

- La soluzione 1 prevede l'utilizzo di
  - Un vettore globale di flag "SC busy"
    - Su tali flag si eseguono un test e una assegnazione
    - La variabile "SC busy" è detta di **lock**
    - Serve a proteggere la SC
  - **Non** garantisce la mutua esclusione in SC

```
while (TRUE) {
  while (flag[j]);
  flag[i] = TRUE;
  SC
  flag[i] = FALSE;
  sezione non critica
}
```

```
while (TRUE) {
  while (flag[i]);
  flag[j] = TRUE;
  SC
  flag[j] = FALSE;
  sezione non critica
}
```

Mutua esclusione: Soluzione 1

- La tecnica fallisce in quanto
  - La variabile di lock viene controllata e modificata mediante due istruzioni a sè stanti
  - Le due istruzioni sono interrompibili (ovvero non sono eseguite in maniera atomica)

```
while (TRUE) {
  while (flag[j]);
  flag[i] = TRUE;
  SC
  flag[i] = FALSE;
  sezione non critica
}
```

```
while (TRUE) {
  while (flag[i]);
  flag[j] = TRUE;
  SC
  flag[j] = FALSE;
  sezione non critica
}
```

Mutua esclusione: Soluzione 1

- Anche se la soluzione fosse corretta
  - I cicli di controllo del flag effettuano una "attesa attiva" (busy waiting)
    - Sprecano CPU time
    - Sono accettabili solo se l'attesa è molto breve
  - L'attesa, ovvero un meccanismo di lock, che utilizza il **busy waiting** viene talvolta denominato **spin lock**

```
while (TRUE) {
  while (flag[j]);
  flag[i] = TRUE;
  SC
  flag[i] = FALSE;
  sezione non critica
}
```

```
while (TRUE) {
  while (flag[i]);
  flag[j] = TRUE;
  SC
  flag[j] = FALSE;
  sezione non critica
}
```

Mutua esclusione: Soluzione 2

- Variabili globali
  - int flag[2] = {FALSE, FALSE};

Scambio di ordine tra test e set nel codice

P_i / T_i

```
while (TRUE) {
   flag[i] = TRUE;
   while (flag[j]);
   SC
   flag[i] = FALSE;
   sezione non critica
}
```

P_j / T_j

```
while (TRUE) {
   flag[j] = TRUE;
   while (flag[i]);
   SC
   flag[j] = FALSE;
   sezione non critica
}
```

Mutua esclusione
Progresso
Attesa definita
Simmetria
?

Mutua esclusione: Soluzione 2

- Variabili globali
  - int flag[2] = {FALSE, FALSE};

P_i / T_i

```
while (TRUE) {
   flag[i] = TRUE;
   while (flag[j]);
   SC
   flag[i] = FALSE;
   sezione non critica
}
```

P_j / T_j

```
while (TRUE) {
   flag[j] = TRUE;
   while (flag[i]);
   SC
   flag[j] = FALSE;
   sezione non critica
}
```

- Progresso **non** garantito
  - P_i e P_j possono rimanere bloccati per sempre (**deadlock** o livelock)

Mutua esclusione: Soluzione 2

- La soluzione 2 tenta di risolvere il problema della soluzione 1 con un approccio simmetrico
  - Invece di testare prima e settare il flag dopo, effettua prima il set e poi il test
  - Può non esserci progresso, ovvero si può presentare deadlock
  - Come la soluzione 1 presenta busy waiting con spin lock

```
while (TRUE) {
  flag[i] = TRUE;
  while (flag[j]);
  SC
  flag[i] = FALSE;
  sezione non critica
}
```

```
while (TRUE) {
  flag[j] = TRUE;
  while (flag[i]);
  SC
  flag[j] = FALSE;
  sezione non critica
}
```

Mutua esclusione: Soluzione 3

- Variabili globali
  - int turn = i;

Oppure
int turn = j;

P_i / T_i

```
while (TRUE) {
  while (turn!=i);
  SC
  turn = j;
  sezione non critica
}
```

P_j / T_j

```
while (TRUE) {
  while (turn!=j);
  SC
  turn = i;
  sezione non critica
}
```

Mutua esclusione
Progresso
Attesa definita
Simmetria
?

Mutua esclusione: Soluzione 3

- Variabili globali
  - int turn = i;

Oppure
int turn = j;

P_i / T_i

```
while (TRUE) {
  while (turn!=i);
  SC
  turn = j;
  sezione non critica
}
```

P_j / T_j

```
while (TRUE) {
  while (turn!=j);
  SC
  turn = i;
  sezione non critica
}
```

- Attesa **non** definita
  - P_i e P_j accedono alla SC in maniera alternata
  - Se P_i (P_j) non è interessato P_j (P_i) va in **starvation**

Mutua esclusione: Soluzione 3

- La soluzione 3 prevede
  - L'uso di una variabile che indica di chi è il "turno"
  - La ME viene garantita mediante cessione del turno
  - La soluzione prevede alternanza e attesa non definita
  - Come le soluzioni 1 e 2 presenta busy waiting con spin lock

```
while (TRUE) {
  while (turn!=i);
  SC
  turn = j;
  sezione non critica
}
```

```
while (TRUE) {
  while (turn!=j);
  SC
  turn = i;
  sezione non critica
}
```

Mutua esclusione: Soluzione 4

- Variabili globali
  - int turn = i;
  - int flag[2] = {FALSE, FALSE};

Oppure
int turn = j;

P_i / T_i

```
while (TRUE) {
   flag[i] = TRUE;
   turn = j;
   while (flag[j] &&
      turn==j);
   SC
   flag[i] = FALSE;
   sezione non critica
}
```

P_j / T_j

```
while (TRUE) {
   flag[j] = TRUE;
   turn = i;
   while (flag[i] &&
      turn==i);
   SC
   flag[j] = FALSE;
   sezione non critica
}
```

Flag a me,
turno a lui,
testo lui

Mutua esclusione
Progresso
Attesa definita
Simmetria
?

Mutua esclusione: Soluzione 4

- Variabili globali
  - int turn = i;
  - int flag[2] = {FALSE, FALSE};

Oppure
int turn = j;

Mutua esculsione?

P_i / T_i

```
while (TRUE) {
  flag[i] = TRUE;
  turn = j;
  while (flag[j] &&
    turn==j);
  SC
  flag[i] = FALSE;
  sezione non critica
}
```

In SC SEE
flag[j]==FALSE OR turn==i

1. P_i e P_j possono entrare insieme?
No, perchè se sono entrambi sul while turn==i oppure turn==j, non entrambi

2. P_i e P_j possono entrare in momenti diversi?
Se P_j è dentro ha fatto richiesta e flag[j]==TRUE, poi però P_i arriva alla richiesta e pone turn=j.
Quindi P_i attenderà

Mutua esclusione: Soluzione 4

- Variabili globali
  - int turn = i;
  - int flag[2] = {FALSE, FALSE};

Oppure
int turn = j;

Progresso?

```
while (TRUE) {          Pi / Ti
  flag[i] = TRUE;
  turn = j;
  while (flag[j] &&
    turn==j);
  SC
  flag[i] = FALSE;
  sezione non critica
}
```

P_i/P_j può rimanere bloccato solo sul ciclo while

1. P_i è sul ciclo e P_j non vuole entrare. Allora flag[j]==FALSE e P_i entra

2. P_i e P_j sono entrambi sul ciclo. Uno dei due entra (vedere condizione di ME)

3. P_i è sul ciclo e P_j esce
P_j mette flag[j]=FALSE e P_i entra

Mutua esclusione: Soluzione 4

- Variabili globali
  - int turn = i;
  - int flag[2] = {FALSE, FALSE};

Oppure
int turn = j;

Attesa definita?

P_i / T_i

```
while (TRUE) {
  flag[i] = TRUE;
  turn = j;
  while (flag[j] &&
    turn==j);
  SC
  flag[i] = FALSE;
  sezione non critica
}
```

P_j è in SC ed è "velocissimo" a ciclare e a richiedere la SC. P_i può attendere per sempre?

P_j assegna FALSE a flag[j] ma subito dopo TRUE. Assegna però turn=i e quindi abilita P_i ad entrare e P_j attende

Mutua esclusione: Soluzione 4

- Variabili globali
  - int turn = i;
  - int flag[2] = {FALSE, FALSE};

Oppure
int turn = j;

Simmetria?

P_i / T_i

```
while (TRUE) {
  flag[i] = TRUE;
  turn = j;
  while (flag[j] &&
    turn==j);
  SC
  flag[i] = FALSE;
  sezione non critica
}
```

P_j / T_j

```
while (TRUE) {
  flag[j] = TRUE;
  turn = i;
  while (flag[i] &&
    turn==i);
  SC
  flag[j] = FALSE;
  sezione non critica
}
```

... Verificata ...
Codici simmetricamente identici

Mutua esclusione: Soluzione 4

- Variabili globali
  - int turn = i;
  - int flag[2] = {FALSE, FALSE};

Oppure
int turn = j;

P_i / T_i

```
while (TRUE) {
  flag[i] = TRUE;
  turn = j;
  while (flag[j] &&
    turn==j);
  SC
  flag[i] = FALSE;
  sezione non critica
}
```

P_j / T_j

```
while (TRUE) {
  flag[j] = TRUE;
  turn = i;
  while (flag[i] &&
    turn==i);
  SC
  flag[j] = FALSE;
  sezione non critica
}
```

- Soluzione corretta
  - Soddisfa tutte le condizione della SC

Mutua esclusione: Soluzione 4

- La prima soluzione software completa è dovuta a G. L. Peterson [1981]
- Garantisce
  - Mutua esclusione
  - Progresso (no deadlock)
  - Attesa limitata (no starvation)
  - Simmetria
- Il P (o T) in attesa è comunque in **busy waiting** su **spin lock**
  - Permane il problema del consumo della risorsa "CPU time"

Conclusioni

- In generale le soluzioni software al problema delle SC risultano complesse e inefficienti
  - L'assegnazione o il controllo di una variabile da parte di un P (o T) è una operazione "invisibile" agli altri P (o T)
  - Le operazioni di controllo e modifica non sono "atomiche", quindi si possono avere reazioni al valore presunto di una variabile invece che a quello reale
  - Le soluzioni per n P (o T) sono complesse
    - McGuire [1972]
    - Lamport [1974]