Soluzioni Software ai problemi di Mutua Esclusione

Soluzioni Software:Primo tentativo

```
Thread1() {
    while(1){
        while( turno == 0 ) ;
        Sezione_Critica1();
        turno = 0;
        Sezione_NON_Critica1();
     }
}
```

```
Thread2(){
      while(1){
        while( turno == 1 ) ;
        Sezione_Critica2();
        turno = 1;
        Sezione_NON_Critica2()
       }
 }
```

Questa soluzione é chiamata 'ALTERNANZA STRETTA CON UNA VARIABILE GLOBALE'. Alcune proprietá di questa soluzione sono:

- soddisfa la muta esclusine
- evita lo stallo tra i processi
- evita la starvation

Problema: Il problema di questa soluzione é che un blocco o un rallentamento in una sezione critica blocca o rallenta l'altro processo.

Soluzioni Software:Secondo tentativo

```
Thread1() {
    while(1){
        while( turno2 == 0 ) ;
        turno1 = 0;
        Sezione_Critica1();
        turno1 = 1;
        Sezione_NON_Critica1();
     }
}
```

```
Thread2(){
      while(1){
        while( turno1 == 0 ) ;
        turno2 = 0;
        Sezione_Critica2();
        turno2 = 1;
        Sezione_NON_Critica2()
       }
 }
```

Le variabili sono inizializzate come segue: turno1=0, turno2=1. Proprietá:

- se una sezione critica si blocca, non viene influenzato l'altro processo
- non c'é stallo
- non c'é starvation

Ma: i due processi possono trovarsi simultaneamente nella sezione critica!!.

Soluzioni Software: Terzo tentativo

```
Thread1() {
    while(1){
        turno1 = 0;
        while( turno2 == 0 ) ;
        Sezione_Critica1();
        turno1 = 1;
        Sezione_NON_Critica1();
     }
}
```

```
Thread2(){
      while(1){
        turno2 = 0;
        while( turno1 == 0 ) ;
        Sezione_Critica2();
        turno2 = 1;
        Sezione_NON_Critica2()
       }
 }
```

Le variabli sono inizializzate come nel caso precedente. Ma: in questa soluzione i processi possono trovarsi in stallo.

Soluzioni Software: Quarto tentativo

Algoritmo di Peterson

```
Thread1() {
  while(1){
    C1 = 1;
    turno = 0;
    while( C2==1 && turno == 0 );
    Sezione_Critica1();
    C1 = 0;
    Sezione_NON_Critica1();
   }
}
```

```
Thread2(){
  while(1){
    C2 = 1
    turno = 1;
    while( C1==1 && turno == 1 );
    Sezione_Critica2();
    C2 = 0;
    Sezione_NON_Critica2();
   }
}
```

Principali proprietá:

- soddisfa a tutte le richieste (cioé mutua esclusione, starvation, stallo)
- la soluzione é estendibile a piú processi

Algoritmo di Peterson

Inseriamo le variabili condivise C in array:

```
Thread1() {
  while(true){
     C[1] = 1;
     t = 0;
     while( C[2] && t == 0 );
     SC1();
     C[1] = 0;
     SNC1();
  }
}
```

```
Thread2(){
  while(true){
     C[2] = 1;
     t = 1;
     while( C[1] && t == 1 );
     SC2();
     C[2] = 0;
     SNC2();
  }
}
```

Consideriamo che il test **C2 == 1** si puó anche scrivere **C2** $\geq$ **C1** .
Analogamente **C1 == 1** si puó anche scrivere **C1** $\geq$ **C2** .

```
Thread1() {
  while(true){
     C[1]=1;
     t = 0;
     while( C[2]≥C[1]&&  t==0 );
     SC1();
     C[1]=0;
     SNC1();
  }
}
```

```
Thread2(){
  while(true){
     C[2]=1;
     t = 1;
     while(C[1]≥C[2] && t == 1 );
     SC2();
     C[2]=0;
     SNC2();
  }
}
```

Algoritmo di Peterson

- I flag 'C' sono per ogni thread e i flag 'turno' per ogni coppia
- L'algoritmo procede per livelli:
  - Per 2 thread, il ogni thread vede se l'altro é nella SC
  - Per 3 thread, abbiamo 3 variabili condivise C e 2 variabili turno
  - Per 3 thread, ognuno vede se gli altri 2 sono nella SC per ciascun valore di turno
  - Per N thread, abbiamo N flag 'C' e N-1 variabili 'turno'
- In generale, lo pseudocodice per N thread é:

```
Pseudocodice Thread i:

for ogni livello j da 1 a N-1 do
      C[i] := j
      turno[j] := i
      for ogni  k da 1 a N do
            if (k!= i)  while(C[k] >= C[i] && turno[j] == i);
      end
end
Sezione Critica i
C[i] := 0
Sezione Non Critica i
```

Attesa attiva!

Pseudocodice dell'Algoritmo di Peterson con 3 Thread

```
Strutture dati inizializzate a  0

Thread P1                Thread P2                Thread P3

C[1]=1;                  C[2]=1;                  C[3]=1;
T[1]=1;                  T[1]=2;                  T[1]=3;
while((C[2] >= C[1]      while((C[1] >= C[2])     while((C[1] >= C[3]
    || C[3] >= C[1])         || C[3] >= C[2])         || C[2] >= C[3])
    && T[1] == 1);           && T[1] == 2);           && T[1] == 3);

C[1]=2;                  C[2]=2;                  C[3]=2;
T[2]=1;                  T[2]=2;                  T[2]=3;
while((C[2] >= C[1]      while((C[1] >= C[2]      while((C[1] >= C[3]
    || C[3] >= C[1])         || C[3] >= C[2])         || C[2] >= C[3])
    && T[2] == 1);           && T[2] == 2);           && T[2] == 3);

SezioneCritica0;         SezioneCritica1;         SezioneCritica2;

C[1]=0;                  C[2]=0;                  C[3]=0;
```

Soluzioni Software: Algoritmo di Lamport

- IDEA: assegno ticket crescenti ai threads
- Far avanzare quello con ticket minore.
- Similitudine con una coda di clienti richiedenti un servizio.

Esempio: supponiamo di avere 3 threads. Ciascuno assegna un ticket C[i] crescente. I ticket sono inizializzati a zero.

```
  Thread 1                  Thread 2                  Thread 3

C[1]=1;                   C[2]=2;                   C[3]=3;
while(C[1]<C[1]);         while(C[1]<C[2]); <--     while(C[1]<C[3]); <--
while(C[2]<C[1]);         while(C[2]<C[2]);         while(C[2]<C[3]); <--
while(C[3]<C[1]);         while(C[3]<C[2]);         while(C[3]<C[3]);
SezioneCritica; <--       SezioneCritica;           SezioneCritica;
C[1]=Max_nthread;         C[2]=Max_nthread;         C[3]=Max_nthread;
SezioneNonCritica;        SezioneNonCritica;        SezioneNonCritica;
```

Mentre, ad esempio, il thread 1 é nella sezione critica, gli altri aspettano.

Max_nthread é il ticket massimo. Problemi: Max_nthread non si conosce! inoltre: problemi di context switch nelle assegnazioni ticket!

Soluzioni Software: Algoritmo di Lamport

Possibile soluzione: invece di usare Max_nthread, usiamo 0:

```
  Thread 1                        Thread 2                          Thread 3

C[1]=1;                         C[2]=2;                           C[3]=3;
while(C[1]!=0&&C[1]<C[1]);      while(C[1]!=0&&C[1]<C[2]); <--    while(C[1]!=0&&C[1]<C[3]); <--
while(C[2]!=0&&C[2]<C[1]);      while(C[2]!=0&&C[2]<C[2]);        while(C[2]!=0&&C[2]<C[3]); <--
while(C[3]!=0&&C[3]<C[1]);      while(C[3]!=0&&C[3]<C[2]);        while(C[3]!=0&&C[3]<C[3]);
SezioneCritica; <--             SezioneCritica;                   SezioneCritica;
C[1]=0;                         C[2]=0;                           C[3]=0;
SezioneNonCritica;              SezioneNonCritica;                SezioneNonCritica;
```

Mentre, ad esempio, il thread 1 é nella sezione critica, gli altri aspettano.

Restano i problemi di context switch nelle assegnazioni ticket!

Algoritmo di Lamport

Per risolvere i problemi di context switch:

```
  Thread 1                          Thread 2                          Thread 3

T[1]=1;                           T[2]=1;                           T[3]=1;
C[1]=1;                           C[2]=2;                           C[3]=3;
T[1]=0;                           T[2]=0;                           T[3]=0;
while(T[1]);                      while(T[1]);                      while(T[1]);
while(C[1]!=0&&C[1]<C[1]);        while(C[1]!=0&&C[1]<C[2]); <--    while(C[1]!=0&&C[1]<C[3]); <--
while(T[2]);                      while(T[2]);                      while(T[2]);
while(C[2]!=0&&C[2]<C[1]);        while(C[2]!=0&&C[2]<C[2]);        while(C[2]!=0&&C[2]<C[3]); <--
while(T[3]);                      while(T[3]);                      while(T[3]);
while(C[3]!=0&&C[3]<C[1]);        while(C[3]!=0&&C[3]<C[2]);        while(C[3]!=0&&C[3]<C[3]);
SezioneCritica; <--               SezioneCritica;                   SezioneCritica;
C[1]=0;                           C[2]=0;                           C[3]=0;
SezioneNonCritica;                SezioneNonCritica;                SezioneNonCritica;
```

Mentre, ad esempio, il thread 1 é nella sezione critica, gli altri aspettano.

Algoritmo di Lamport

Assegnazione dei ticket:

```
  Thread 1                          Thread 2                          Thread 3

T[1]=1;                           T[2]=1;                           T[3]=1;
C[1]=1+max(C[1],C[2],C[3]);       C[2]=1+max(C[1],C[2],C[3]);       C[3]=1+max(C[1],C[2],C[3]);
T[1]=0;                           T[2]=0;                           T[3]=0;
while(T[1]);                      while(T[1]);                      while(T[1]);
while(C[1]!=0&&C[1]<C[1]);        while(C[1]!=0&&C[1]<C[2]); <--    while(C[1]!=0&&C[1]<C[3]); <--
while(T[2]);                      while(T[2]);                      while(T[2]);
while(C[2]!=0&&C[2]<C[1]);        while(C[2]!=0&&C[2]<C[2]);        while(C[2]!=0&&C[2]<C[3]); <--
while(T[3]);                      while(T[3]);                      while(T[3]);
while(C[3]!=0&&C[3]<C[1]);        while(C[3]!=0&&C[3]<C[2]);        while(C[3]!=0&&C[3]<C[3]);
SezioneCritica; <--               SezioneCritica;                   SezioneCritica;
C[1]=0;                           C[2]=0;                           C[3]=0;
SezioneNonCritica;                SezioneNonCritica;                SezioneNonCritica;
```

Mentre, ad esempio, il thread 1 é nella sezione critica, gli altri aspettano.

Attenzione: context swith nelle assegnazioni dei ticket possono far si che due ticket sono uguali!

Algoritmo di Lamport

Per risolvere il problema dei ticket uguali, si puó complicare il confronto tra ticket.
Introduciamo l'operatore di confronto tra coppie di interi: (a,b) $<$ (c,d) se a $<$ c e, se a $=$ c, b $<$ d. In conclusione:

```
  Thread 1                          Thread 2                          Thread 3

T[1]=1;                           T[2]=1;                           T[3]=1;
C[1]=1+max(C[1],C[2],C[3]);       C[2]=1+max(C[1],C[2],C[3]);       C[3]=1+max(C[1],C[2],C[3]);
T[1]=0;                           T[2]=0;                           T[3]=0;
while(T[1]);                      while(T[1]);                      while(T[1]);
while(C[1]!=0 &&                  while(C[1]!=0 &&                  while(C[1]!=0 &&
     ((C[1],1)<(C[1],1)));             ((C[1],1)<(C[2],2))); <--         ((C[1],1)<(C[3],3))); <--
while(T[2]);                      while(T[2]);                      while(T[2]);
while(C[2]!=0 &&                  while(C[2]!=0 &&                  while(C[2]!=0 &&
     ((C[2],2)<(C[1],1)));             ((C[2],2)<(C[2],2)));             ((C[2],2)<(C[3],3))); <--
while(T[3]);                      while(T[3]);                      while(T[3]);
while(C[3]!=0 &&                  while(C[3]!=0 &&                  while(C[3]!=0 &&
    ((C[3],3)<(C[1],1)));             ((C[3],3)<(C[2],2)));             ((C[3],3)<(C[3],3)));
SezioneCritica; <--               SezioneCritica;                   SezioneCritica;
C[1]=0;                           C[2]=0;                           C[3]=0;
SezioneNonCritica;                SezioneNonCritica;                SezioneNonCritica;
```

Mentre, ad esempio, il thread 1 é nella sezione critica, gli altri aspettano.

Algoritmo di Lamport per N threadi

```
// valori iniziali variabili condivise
  T: array [1..N] of bool = {false};
  C: array [1..N] of integer = {0}; //tickets
  lock(integer i) {
     T[i] = true;
     C[i] = 1 + max(C[1], ..., C[N]);
     T[i] = false;
     for (j = 1; j <= N; j++) {
         // Attendi finche' il thread j riceve il suo ticket
         while (T[j]) {};
         // Attendi la fine dei thread con ticket minore:
        while ((C[j] != 0) && ((C[j], j) < (C[i], i))) {};
    }
}
unlock(integer i) {
    C[i] = 0;
}
Thread(integer i) {
    while (true) {
        lock(i);
        // Sezione critica
        unlock(i);
        // Sezione NON critica
    }
}
```