

```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAXPAROLA 30
#define MAXRIGA 80

int main(int argc, char *argv[])
{
    int freq[MAXPAROLA]; /* vettore di contatori
delle frequenze delle lunghezze delle parole */
    char riga[MAXRIGA];
    int i, inizio, lunghezza;
    FILE *f;

    for(i=0; i<MAXPAROLA; i++)
        freq[i]=0;

    if(argc != 2)
    {
        fprintf(stderr, "ERRORE: serve un parametro con il nome del file\n");
        exit(1);
    }
    f = fopen(argv[1], "r");
    if(f==NULL)
    {
        fprintf(stderr, "ERRORE: impossibile aprire il file %s\n", argv[1]);
        exit(1);
    }

    while( fgets( riga, MAXRIGA, f ) != NULL )
```

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};`

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

```
while (TRUE) { Pj / Tj  
    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};`

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

```
while (TRUE) { Pj / Tj  
    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"
 - Il test e il set di tali flag
- ❖ **Non** garantisce la mutua esclusione in SC
- ❖ 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**)

Mutua esclusione: Soluzione 1

- ❖ La variabile di flag "SC busy" è detta di **lock**
 - Serve a proteggere la SC
- ❖ 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
- ❖ Un'attesa, ovvero un meccanismo di lock, che utilizza il busy waiting viene talvolta denominato **spin lock**

Mutua esclusione: Soluzione 2

❖ Variabili globali

➤ `int flag[2] = {FALSE, FALSE};`

Scambio di ordine tra
test e set nel codice

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

P_i / T_i

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

P_j / T_j

Mutua esclusione
Progresso
Attesa definita
Simmetria

?

Mutua esclusione: Soluzione 2

❖ Variabili globali

➤ `int flag[2] = {FALSE, FALSE};`

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

```
while (TRUE) { Pj / Tj  
    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 e settare il flag prima effettua 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

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

Mutua esclusione: Soluzione 4

❖ Variabili globali

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

Oppure
`int turn = j;`

```

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

```

P_i / T_i

```

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

```

P_j / T_j

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
esclusione?

```

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

P_i / T_i

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

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

Se però P_j è in SC e P_i arriva dopo può entrare?
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) {
    flag[i] = TRUE;
    turn = j;
    while (flag[j] &&
           turn==j);
    SC
    flag[i] = FALSE;
    sezione non critica
}
    
```

P_i / T_i

P_i/P_j può rimanere bloccato solo sul ciclo while

Se P_i è sul ciclo e P_j non vuole entrare, allora `flag[j]==FALSE` e P_i entra

Se P_i è sul ciclo e P_j è anch'esso sul while, uno dei due entra (vedere cond. ME)

Se 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?

```

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

```

P_i / T_i

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?

```

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

```

P_i / T_i

```

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

```

P_j / T_j

... Verificata ...
Codici simmetricamente identici

Mutua esclusione: Soluzione 4

❖ Variabili globali

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

Oppure
`int turn = j;`

```

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

```

P_i / T_i

```

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

```

P_j / T_j

❖ Soluzione corretta

- Soddisfa tutte le condizone 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]