Soluzioni architetturali ai problemi di Mutua Esclusione

Soluzioni architetturali

Supponiamo di avere una variabile condivisa tra i thread, *lock*.
Inizializzamo $lock = 0$
Possiamo usare *lock* per risolvere la mutua esclusione? Vediamo...

```
Thread 1                    Thread 2                    Thread 3

--                          --                          --
--                          --                          --
while(lock==1);             while(lock==1);             while(lock==1);
lock=1;                     lock=1;                     lock=1;
SezioneCritica;             SezioneCritica;             SezioneCritica;
lock=0;                     lock=0;                     lock=0;
--                          --                          --
--                          --                          --
```

Supponiamo che Thread 1 sia il primo.
Ahimé, basta un context switch dopo il primo while a negare la mutua esclusione!

Questa nonsoluzione crea altre SezioniCritiche oltre a quelle che bisognerebbe risolvere!

Soluzioni architetturali

Supponiamo di avere, oltre alla variabile *lock* inizializzata a 0, una funzione chiamata TAS (da TestAndSet):

```
int TAS(){
    int temp=lock;
    lock=1;
    return(temp);
}
```

Possiamo usare TAS per risolvere la mutua esclusione? Vediamo...

```
Thread 1                    Thread 2                    Thread 3
--                          --                          --
while(TAS());               while(TAS());               while(TAS());
SezioneCritica;             SezioneCritica;             SezioneCritica;
lock=0;                     lock=0;                     lock=0;
--                          --                          --
```

Questa soluzione ha gli stessi problemi di mutua esclusione della precedente nonsoluzione. Tuttavia ha il vantaggio che TAS potrebbe essere realizzata con un unica istruzione assembler. Se l'istruzione non fosse interrompibile, risolverebbe il problema! $\rightarrow$ Istruzioni ATOMICHE!

Soluzioni architetturali

COME FACCIO AD AVERE ISTRUZIONI NON INTERROMPIBILI?? RICORSO ALLE ARCHITETTURA DEI CALCOLATORI (MICROPROGRAMMA!!)
Esaminiamo alcune varianti: l'operazione TAS:

```
int TAS(){
    int temp=lock;
    lock=1;
    return(temp);
}
```

puó anche essere vista come:

```
int testset()
{
  if (lock==0) {
    lock=1;
    return 1;
  } else {
    return 0;
  }
}
```

Questa funzione dá l'idea di costruire un'altra funzione. Vediamo:

Soluzioni architetturali

Supponiamo di avere, oltre alla variabile condivisa *lock*, anche una funzione CAS:

```
int CAS(int test, int new)
{
    int temp=lock;
    if(temp==test)
            lock=new;
    return(temp);
}
```

CAS puó essere usata nel seguente modo:

```
Thread 1
--
while(CAS(0,1)==1);
SezioneCritica;
lock=0;
--
```

```
Thread 2
--
while(CAS(0,1)==1);
SezioneCritica;
lock=0;
--
```

```
Thread 3
--
while(CAS(0,1)==1);
SezioneCritica;
lock=0;
--
```

Analizzando vari possibili contextswitch, vediamo che se CAS fosse non interrompibile, risolverebbe la Mutua Esclusione!

Soluzioni architetturali

Altra variante: Supponiamo di avere una funzione SWAP:

```
int SWAP(int test, int new)
{
    int temp=new;
    new=test;
    temp=test;
}
```

ATTENZIONE: *test* e *new* sono passate PER INDIRIZZO!!
ATTENZIONE: *test* é una variabile locale, *new* é una variabile CONDIVISA!
SWAP puó essere usata nel seguente modo: inizializziamo $new = 0$.

```
Thread 1
--
a=1;
do
   scambia(a,b);
while(a!=0);
SezioneCritica;
b=0;
--
```

```
Thread 2
--
a=1;
do
    scambia(a,b);
while(a!=0);
SezioneCritica;
b=0;
--
```

```
Thread 3
--
a=1;
do
    scambia(a,b);
while(a!=0);
SezioneCritica;
b=0;
--
```

Soluzioni architetturali

Attenzione: TAS, CAS, SWAP sono istruzioni presenti nei processori!
Attenzione: TAS, CAS, SWAP sono descritti con psudocodice.

COME POTREBBERO ESSERE TRADOTTE IN JAVA??? COME POSSONO ESSERE TRADOTTE IN C???