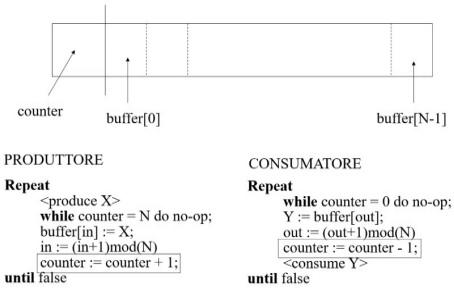
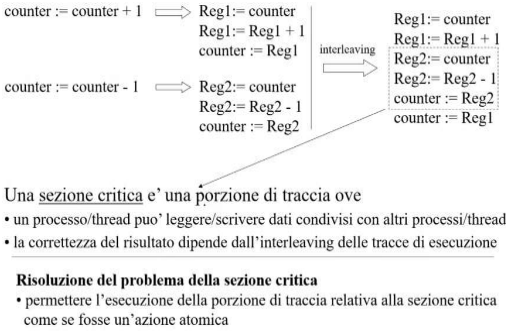
Sincronizzazione 1 Intro + Sezione critica

giovedì 20 novembre 2025 18:59

Prima di addentarci nel discorso, facciamo un riepilogo sulla funzione **printf()** : questa libreria internamente utilizza un meccanismo si sincronizzazione , mantenendo la coerenza dei dati se in presenza di più thread. Lo stesso discorso vale per gli address space . Andiamo ora a vedere la **sezione critica** di questa applicazione : studiamo il problema classico del **produttore consumatore** :



Si supponga che l'area condivisa sia una shared memory ; per contatore si intende quell'intero che rappresenta le informazioni scritte ma non lette . **Sia lettura che scrittura usano il buffer circolare (si parte da 0 fino a N-1)**. Questo algoritmo ha un enorme problema di sincronizzazione : attenzione all'isa della macchina , il quale ci porta a ciò:



Ovvero la perdita dei dati avendo così inconsistenza dei dati. Quindi per ovviare a questo problema le istruzioni vengono eseguite come unica attività atomica. Vediamolo in dettaglio :

```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

#define SIZE (100000)

volatile long v[SIZE] = {[0 ... (SIZE-1)] -1};

volatile long counter = 0;


void * producer(void* dummy){

        long data = 0;
        long my_index = 0;

        printf("ready to produce\n");

retry:
        while(counter < SIZE){
                v[my_index] = data;
                my_index = (my_index+1)%SIZE;
                data++;
                counter++;
        }
        goto retry;

}
```

```c
void * consumer(void* dummy){

        long data = 0;
        long my_index = 0;
        long value;

        printf("ready to consume\n");

retry:
        while(counter > 0){
                value = v[my_index];
//              printf("got value %ld\n",value);
                if(value != data){
                        printf("consumer: synch protocol broken at expected value: %ld - real is %ld!\n",data,value);
                        exit(EXIT_FAILURE);

                };
                data++ ;
                my_index = (my_index+1)%SIZE;
                counter--;

        }
        goto retry;

}
```

```c
int main(int argc, char** argv){

        pthread_t prod, cons;
        pthread_create(&cons,NULL,consumer,NULL);
        sleep(1);
        pthread_create(&prod,NULL,producer,NULL);

        pause();

}
```

Eseguendolo si ha che il valore del contatore non è gestito in modo corretto : si è perso un decremento.

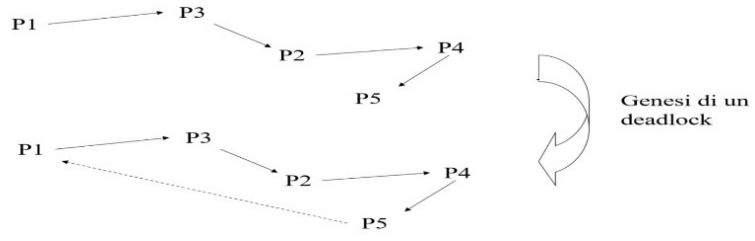
```
ready to consume
ready to produce
consumer: synch protocol broken at expected value: 88 - real is -1!
```

Quindi vediamo ora come bypassare questo problema :

1. **Mutua esclusione**
   a. Se un thread accede a questa sezione, nessun'altro thread vi può accedere
   b. Esecuzione intero blocco istruzioni
2. **Progresso**
   a. Un thread accede alla sezione critica se nessun'altro vi accede se nessun ritardo dovuti ad altre esecuzioni di thread
3. **Attesa limitata**
   a. La sezione critica viene eseguita da unico thread in un tempo limitato (non ci sono stalli o starvation)
   b. Vediamo in dettaglio lo stallo:

Un insieme di processi P1, ..., Pn e' coinvolto in uno stallo se ognuno dei processi e' in attesa (attiva o passiva) di un evento che puo' essere causato solo da un altro dei processi dell'insieme

i.



   ii. Quindi lo stallo è una catena di processi/thread bloccati

Quindi gi approcci risolutivi sono **algoritmi di mutua esclusione , approcci RMW (read write modify) oppure mutex/semafori**.