Addendum sui SEMAFORI

EM-DIA

Cosa sono i semafori?

- I semafori sono primitive fornite dal sistema operativo per permettere la sincronizzazione tra processi e/o thread.

Operazioni sui semafori

- In generale sono tre le operazioni che vengono eseguite ATOMICAMENTE da un processo su un semaforo:
  - Creazione (sem_init) o collegamento (shm_open)
  - Wait. Attesa su di un semaforo dove si verifica il valore del semaforo: system call `sem_wait(semaforo)`

    while(sem_value<=0) aspetta;

    sem_value--;

  - Post. Incremento del semaforo: system call `sem_post(semaforo)`

    sem_value++;

Cosa sono i mutex?

- Un mutex è un semaforo binario che serve per la protezione delle sezioni critiche:
  - variabili condivise modificate da più processi o thread
  - solo un processo o thread alla volta può accedere ad una risorsa protetta da un mutex

- Il valore di un semaforo binario può essere 0 (occupato) oppure 1 (libero

Cosa sono i mutex?

- Pensiamo ai mutexcome a delle serrature:
  - il primo processo o thread che ha accesso alla coda dei processi o thread lascia fuori gli altri fino a che non ha portato a termine il suo compito.
- I processi o threads inseriscono un mutex prima delle sezioni di codice nelle quali vengono condivisi i dati.

Semafori contatori

- Valore da 0 in su
- Usati per soluzioni semaforiche, non di mutua esclusione
- Realizzabili con semafori binari
- Quando sono = 0 bloccano il semaforo, altrimenti contano eventi

Soluzioni semaforiche e di mutua esclusione a problemi di sincronizzazione

- Un problema di sincronizzazione
  - Due variabili, *na* e *nb*, vengono incrementate continuamente da due processi concorrenti, P1 e P2.

    Si vuole fare in modo che

    $$na <= nb+4$$
  - La situazione è dunque:

P1

```
while(true){
    na++;
}
```

P2

```
while(true){
    nb++;
}
```

Una soluzione di mutua esclusione al problema

- Mutex deve essere condiviso tra I processi

P1

```
while(true){
    if(na >= nb+4)
       sem_wait(mutex);

    na++;
}
```

P2

```
while(true){
    nb++;

    if(na<nb+4)
       sem_post(mutex)
}
```

Una soluzione semaforica al problema

- *Sem* deve essere condiviso tra i processi
- *Sem* viene inizializzato come: *Sem = nb +4 - na*

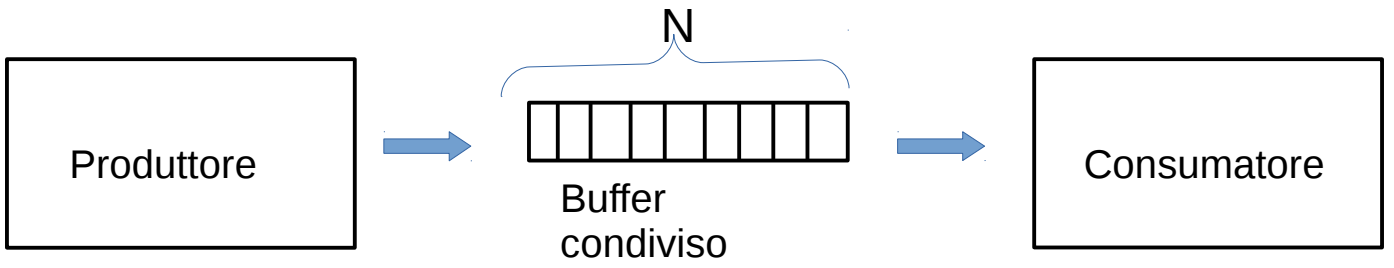
P1

```
while(true){
    sem_wait(Sem);
    na++;
}
```

P2

```
while(true){
    nb++;
    sem_post(Sem)
}
```

Mutua esclusione nel produttore-consumatore

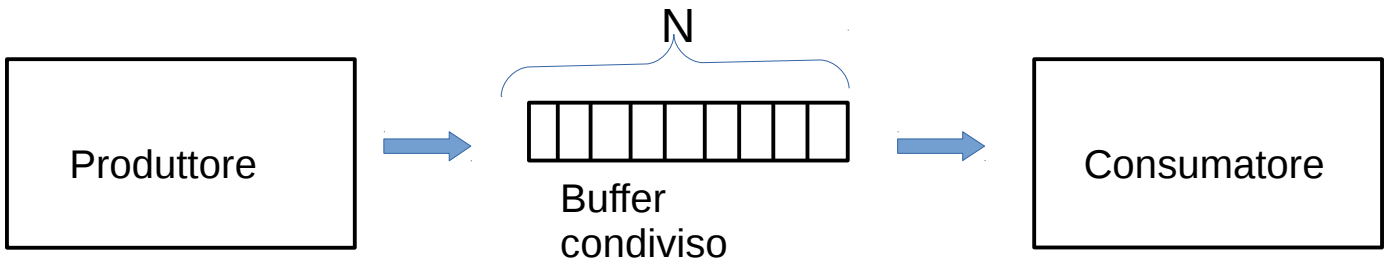


- Primo tentativo

```
if ((pid = fork()) == 0){ /* consumatore*/
        for (i = 0; i < 20; i++)  {
            if (*in == *out) sem_wait(mutex);
            value = buffer[*out]; /* leggo il buffer */
            *out = (*out + 1) % BUF_SIZE;
            printf ("\tleggo buf[%d] == %d\n", *out, value);
           }
}
else{   /* produttore */
        for (j = 0; j < 20; j++)   {
            if ((*in + 1) % BUF_SIZE == *out) sem_wait(mutex);
            buffer[*in] = j;  /* scrivo il buffer */
            *in = (*in + 1) % BUF_SIZE;
            printf ("ho scritto  %d  in buf[%d]\n", j, *in);
        }
        wait (pid);
}
```

- Deadlock. Come si può eliminare ?

Mutua esclusione nel produttore-consumatore



- Algoritmi
- n=nr elementi
- Come scrivere
  - estrai, inserisci?

Processo Produttore:

```
while(true){
     el=produci();

     if(n==N) sem_wait(mutex1); //se il buffer e' pieno, aspetta
     inserisci(el);
     n++;
     if(n==1) sem_post(mutex2); //c'è un elemento: sblocca mutex2
     Inserisci(el);
}
```

Processo Consumatore

```
while(true){
    if(n==0) sem_wait(mutex2);   //se il buffer e' vuoto, aspetta
    el=estrai();
    n--;
    if(n==N-1) sem_post(mutex1); //c'è un posto libero: sblocca il mutex
    consuma(el);
}
```