[Esercizio3 (P&C).pdf](https://drive.google.com/open?id=19vHQOw-nLzB0249agEZzI5DIwinVQ1cR)

Si consideri il classico problema dei produttori e consumatori, con il buffer implementato con un array di interi di dimensione 100.

Si assuma, per semplicita’, che gli interi prodotti e consumati siano >=0. Si considerino le seguenti condizioni aggiuntive:

1. Gli interi pari possono occupare solo le posizioni da 0 a 49;
2. Gli interi dispari possono occupare solo le posizioni da 50 a 99;
3. Vi e’ una nuova categoria di processi, i processi ConsPari, che consumano tutti gli interi pari, se l’array contiene almeno 20 interi pari;

Quanto un processo tenta di effettuare un’operazione al momento non consentita (per esempio produrre un intero che il buffer non può al momento ospitare), il processo deve essere messo in attesa.

Programmare il sistema sfruttando i semafori con la semantica tradizionale.

Appunti iniziali:

* Buffer di 100 interi >=0
* Interi pari -> posizioni 0-49
* Interi dispari -> posizioni 50-99
* Produttore, Consumatore, ConsPari

## Semafori:

emptyPari=50 //

emptyDispari=50 //

mutexPari=1 //

mutexDispari=1 //

semProd=0

semCons=0

semConsP=0

## Variabili:

numPari=0 // Numero di interi pari

numDispari=0 // Numero di interi dispari

i=0 // Contatore pari

j=0 // Contatore dispari

k=0 // Contatore per ciclo in cui vengono svuotati gli indici pari

waitProd=0 // Numero di produttori in attesa

waitCons=0 // Numero di consumatori in attesa

waitConsP=0 // Numero di ConsPari in attesa

Supponiamo che l’array e’ inizializzato con valori negativi

# Produttore

while(true) {

…

item=...;

if(item%2==0) { // Item pari

wait(mutexPari);

if(numPari==50) { // Se int pari pieni

waitProd++;

wait(emptyPari);

signal(mutexPari);

}

while(buffer[i]<0)

i=(i+1)%50;

buffer[i]=item;

i=(i+1)%50;

numPari++;

wait(emptyPari);

signal(mutexPari);

if(numPari>=20 && waitConsP>0) {

waitConsP- -;

signal(

}

}

else { // Item dispari

wait(mutexDispari);

if(

}

}

# Consumatore

while(true) {

wait(mutexCons);

if((numPari+numDispari) == 0) { // Se array vuoto

waitCons++;

wait(semCons);

}

while(buffer[k]<0)

k=(k+1)%100;

item=buffer[k];

k=(k+1)%100;

signal(mutexCons);

}

# ConsPari

while(true) {

}

Con Depa:

Si assume l’array inizializzato con numero negativi.

## Variabili:

* contPari=0
* contDispari=0
* i=0 // Non va scritto
* j=0 // Non va scritto
* k=0 // Non va scritto
* buffer[100] // Non va scritto
* waitPari=0
* waitDispari=0
* waitCons=0
* waitConsPari=0

## Semafori:

* semPari=0
* semDispari=0
* semCons=0
* semConsPari=0
* mutex=1
* emptyPari
* emptyDispari

# Produttore

while(true) {

…

int item=...;

wait(mutex);

if(item%2==0) { // Se pari

if(contPari==50) { // Se non ci sono posizioni libere per pari

waitPari++;

signal(mutex);

wait(semProd);

wait(mutex);

}

buffer[i]=item;

i=(i+1)%50;

contPari++;

if(contPari>=20 && waitConsPari>0) {

waitConsPari--;

signal(semConsPari);

}

signal(mutex);

}

else { // Se dispari

if(contDispari==50) { // Se non ci sono posizioni libere per dispari

waitDispari++;

signal(mutex);

wait(semProd);

wait(mutex);

}

buffer[j]=item;

j=((j+1)%50)+50;

contDispari++;

signal(mutex);

}

wait(mutex);

if((contPari + conDispari>0) && waitCons>0) { // Se buffer non vuoto e c'è un consumatore in attesa, rilascia uso del cons

waitCons--;

signal(semCons);

}

…

}

# Consumatore

while(true) {

…

wait(mutex);

if(contPari + contDispari == 0) { // Se array vuoto

waitCons++;

signal(mutex);

wait(semCons);

wait(mutex);

}

while(buffer[k]<0)

k=(k+1)%100;

item=buffer[k];

buffer[k]=-1; // Lo “consumiamo” mettendo valore negativo

k=(k+1)%100;

if(item%2==0) { // Se pari

contPari--;

if(waitPari>0) { // Se ci sono produttori pari in attesa

waitPari--;

}

}

}

# ConsPari

while(true) {

}