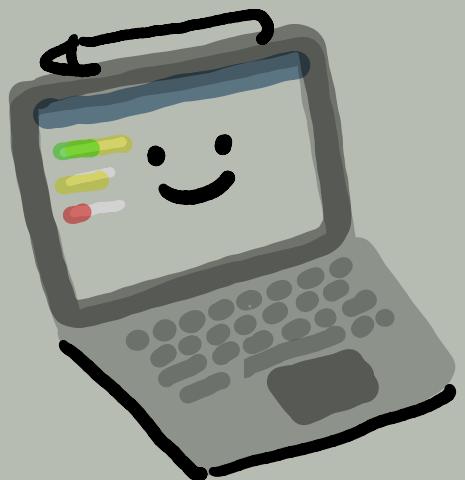


Sistemi Operativi



carmine coppola /

Sincronizzazione

- In un ristorante self-service, i clienti, dopo aver mangiato, dispongono i vassoi in M contenitori, ognuno di K ripiani.
- Periodicamente, un addetto sceglie un contenitore tra quelli in cui ci sono più ripiani liberi, lo svuota, lava i piatti e riporta il contenitore in sala.

Cliente :

- ① Arriva
- ② Prende il vassoio da contenitore puliti
- ③ Mangia
- ④ Posa il vassoio nel contenitore sporchi
- ⑤ Lascia il negozio

Addetto :

- ① Aspetta il cliente
- ② Aspetta vassoio
- ③ Prende vassoio dal contenitore sporchi
- ④ lava vassoio
- ⑤ Posa vassoio nel contenitore puliti

Var

semaforo_contatore : vassoio_disponibile = M

semaforo_contatore : vassoio_sporco = 0

semaforo : cliente

semaforo : addetto

begin

signal (cliente)

lock (my_mutex)

if (vassoio_disponibile > 0) {

unlock (my_mutex)

wait (vassoio_disponibile) // decremento il num.
di vassoi puliti

lock (my_mutex)

{ prende il vassoio}

scelta_cliente = vassoi [rand (indice_v)]

vassoi [indice_v] --

unlock (my_mutex)

mangia ()

lock (my_mutex)

{ posa il vassoio}

vassoi [indice_v] ++

unlock (my_mutex)

Signal (vassoio_sporco)

signal (addetto)

z { lascia il negozio}

else

wait (cliente) // Non ci sono vassoi disponibili

Cliente

vassoi : array of [K]

scelta_cliente : var

scelta_addetto : var

my_mutex : MUTEX = 1

repeat

lock (my_mutex)

if (diente == 0)

unlock (my_mutex)

wait (addetto) // Aspetto che il diente mangi
ci sia vassoio sporco

else

wait (vassoio_sporco) // decremento il num.
di vassoi sporchi

lock (my_mutex)

{ prende ripiano con meno vassoi}

for i to K

scelta_addetto = min (i)

vassoi [i] --

unlock (my_mutex)

lava ()

wait (cliente) // Sto posando i vassoi puliti

lock (my_mutex)

{ poso i vassoi lavati}

for i to K // Segnalo e incremento
i vassoi puliti

vassoi [indice_v] ++

signal (vassoi_disponibili)

unlock (my_mutex)

signal (cliente) // Sblocco i clienti

Forever

Addetto

Un autobus è dotato di **N posti a sedere** ed **M posti in piedi**. L'accesso dei passeggeri all'autobus avviene mediante **tre entrate**:

1. Una posizionata in **testa**;
2. Una al **centro**;
3. Una in **coda**;

Tutte le entrate possono essere utilizzate sia per **salire** che per **scendere** dall'autobus.

Supposizioni: si sale/scende uno alla volta dando **precedenza a chi scende** su tutte le porte.

Salita :

- ① Scelgo l'entrata.
- ② Controllo se c'è qualcuno in uscita.
↳ Attendo che esca.
- ③ Salgo
- ④ Controllo se ci sono posti.
↳ No Scendo
- ⑤ Controllo i posti a sedere.
↳ Si Mi siedo
- ⑥ Controllo se ci sono posti in piedi.
↳ Si Mi siedo
- ⑦ Scendo

Discesa :

- ① Segnalo che il posto è libero
- ② Scelgo l'uscita.
- ③ Esco dall'autobus

Vaz:

```
semaforo . contatore : posti . s = N  
semaforo . contatore : posti . inP = M  
semaforo : entrata = 1  
Semaforo : discesa = 0
```

begin

```
scelta . s = rand (1, 3) // perché 3 sono le entrate
```

```
lock (my . mutex)
```

```
if (discesa == 1)
```

```
unlock (my . mutex)
```

```
wait (entrata)
```

```
else {
```

```
lock (my . mutex)
```

```
if (posti . s > 0) {
```

```
unlock (my . mutex)
```

```
wait (posti . s)
```

```
lock (my . mutex)
```

```
posti . s [i] --
```

```
{ Mi siedo }
```

```
unlock (my . mutex)
```

```
}
```

```
lock (my . mutex)
```

```
else if (posti . inP > 0) {
```

```
unlock (my . mutex)
```

```
wait (posti . inP)
```

```
lock (my . mutex)
```

```
posti . inP [i] --
```

```
{ Sto in Piedi }
```

```
unlock (my . mutex)
```

```
}
```

```
else
```

```
{ Scendo }
```

```
scelta . s : integer = 0  
scelta . d : integer = 0  
my . posto . var = {"Seduto", "In . Piedi"}  
my . mutex : MUTEX = 1
```

begin

```
if (my . posto == "Seduto") {
```

```
signal (posti . s)
```

```
posti . s ++
```

```
wait (entrata)
```

```
lock (my . mutex)
```

```
scelta . d = rand ()
```

```
{ scende }
```

```
unlock (my . mutex)
```

```
Signal (discesa)
```

```
Signal (entrata)
```

```
}
```

```
else {
```

```
signal (posti . inP)
```

```
posti . inP ++
```

```
wait (entrata)
```

```
lock (my . mutex)
```

```
scelta . d = rand ()
```

```
{ scende }
```

```
unlock (my . mutex)
```

```
Signal (discesa)
```

```
Signal (entrata)
```

```
}
```

Discesa

Salita

Un gruppo di amici festeggia in un pub con B bottiglie di birra.

Il garzone del pub aggiunge una bottiglia di birra ogni volta che riscontra personalmente che una bottiglia è vuota. Egli agisce in modo prioritario rispetto agli amici.

Gli amici hanno concordato di accedere alle B bottiglie di birra per NON più di M volte consecutive ciascuno, tra due consecutive sostituzioni di bottiglia effettuate dal garzone.

(Si supponga per semplicità che ogni volta che un amico accede alle bottiglie riempie per intero il proprio boccale).

Garzone :

- ① Controlla se ci sono bottiglie vuote
↳ sì
- ② Aggiunge bottiglie di birra

Amici :

- ① Non utilizzano la bottiglia più di M volte
- ② Riempiano il proprio boccale
- ③ Bevano birra

Vari:

bottiglie_full : semaforo_cont := N

garzone : semaforo := 1

amici : semaforo := 0



Garzone :

repeat

lock(my_mutex)

if(bottiglie_full > 0)

// Se le bottiglie ci sono allora

unlock(my_mutex)

il garzone non deve aggiungere

wait(garzone)

la bottiglia, quindi lo blocco

else

wait(amici)

// Fermo gli amici poiché
deve sostituire la bottiglia
(aggiungere)

lock(my_mutex)

aggiunge_bottiglia();

signal(bottiglie_full) // Incrementa le bottiglie

sostituzione++

signal(sostit)

unlock(my_mutex)

signal(amici) // riattiva gli amici

Forever

Amici :

if(accesso < M)

wait(bottiglie_full)

lock(my_mutex)

if(sostituzioni < 2)

riempio_boccale()

unlock(my_mutex)

bevo()

signal(garzone)

else

unlock(my_mutex)

wait(sostit)

else

wait(amici)

Se ho utilizzato meno di M volte
la bottiglia proceedo a riempire il
boccale, derementando la bottiglia piena
e incrementando il numero di accessi
alla bottiglia (in M.E.).

Dopo aver bevuto attivo il garzone.

Altrimenti se ho utilizzato più di M
volte la bottiglia non bevo e fermo
gli amici.

In una pizzeria, **due pizzaioli** sfornano **continuamente** pizze che pongono di volta in volta in uno di N piatti disponibili su un bancone.

I piatti sono prelevati da M ($M < N$) camerieri che provvedono a servirle ai tavoli. Ciascun cameriere può portare **2 pizze per volta**, se disponibili, ma, **in caso di affollamento**, i camerieri **preleveranno 3 pizze per volta**. Fornire una soluzione con semafori e discutere eventuali problemi di starvation/deadlock

Pizzaioli (2)

- ① Sforna pizza
- ② Controllano se sono disponibili piatti disponibili sul bancone
- ③ Pongono pizze in N piatti

vaz:

cliente sem - cont := 0

piatti - cameriere : sem - cont := N

piatti - pizzaiolo : sem - cont := N

my - mutex : MUTEX := 1

piatti : array of [N]

num - p : integer := 0

num - pizze : integer := 0

indice - piatti : integer := 0

CLIENTE :

signal (cliente)

consuma ()

wait (cliente)

CAMERIERE

lock (my - mutex)

if (cliente > 10)

// controllo se c'è folla

unlock (my - mutex)

for (i = 0 to 3) {

wait (piatti - cameriere)

// controllo se ci sono i piatti e

lock (my - mutex)

// decremento il numero di piatti

num - p = piatti [indice - piatti]

// mi salvo il numero di piatti pres

indice - piatti --

// e decremento l'indice

unlock (my - mutex)

}

sewo_ai_tavoli ()

for (i = 0 to 3)

// segnalo la nuova disponibilità

signal (piatti - pizzaiolo)

// dei piatti per il pizzaiolo

else

for (i = 0 to 2) {

// qualora non c'è folla

wait (piatti - cameriere)

// faccio lo stesso procedimento

lock (my - mutex)

// per 2 piatti

num - p = piatti [indice - piatti]

indice - piatti --

unlock (my - mutex)

}

sewo_ai_tavoli ()

for (i = 0 to 2)

signal (piatti - pizzaiolo)

Pizzaiolo .

num - pizze = sforna - pizza ()

wait (piatti - pizzaiolo)

lock (my - mutex)

piatti [ind - p ++] = num - pizze

unlock (my - mutex)

signal (piatti - cameriere)

Sforna le pizze e le associo

ad una variabile, dopodiché

controllo se ci sono piatti sul bancone,

entro in mutua esclusione per inserire

nell'array e segnalo che la pizza è

stata impiazzata.

Un sistema è composto di un buffer di N posizioni, $2 \cdot N$ processi attivi ed un **processo coordinatore**. Il **processo coordinatore** estrae uno dopo l'altro in maniera casuale i numeri da 1 a N e ad ogni estrazione i processi competono per aggiudicarsi l'accesso alla posizione corrispondente del buffer, scrivendone il proprio **PID**. Un processo che ha scritto il proprio **PID** nel buffer NON partecipa più alla contesa. Quando tutte le posizioni del buffer sono state assegnate, il coordinatore si **alterna** con l'ultimo processo che ha avuto accesso al buffer nel leggere e stampare, uno alla volta, il contenuto del buffer.

Coordinatore .

- ① Estrae in maniera casuale i numeri da 1 a N
- ② Attende che tutte le posizioni del buffer siano assegnate
- ③ Si alterna con l'ultimo processo per stampare e leggere il contenuto del buffer

Var:

```
estratto : semaforo := 0
coordinatore : semaforo := 1
turno_coordinatore : semaforo := 1
```

```
turno_ultimo_processo : se := 0
buffer : array of [N]
buffer_full : boolean := false
```

```
ind_stampa : integer := 0
processo : integer := 0
indice : integer := 0
```

```
main () {
    crea_processo(coordinatore)
    for (i = 0 to 2 * N)
        crea_processo(processo_attivo)
}
```

Coordinatore :

```
ind_num = estrai_num(1, N)
signal(estratto)
Wait(coordinatore)
if (buffer_full == true) {
    for (ind_stampa = 0 to N) {
        wait(turno_coordinatore)
        stampo(buffer[ind_stampa])
        signal(turno_ultimo_processo)
    }
}
```

Estraggo N numeri, la mia funzione genera numeri diversi tra di loro. Segnalo al processo che i numeri siano stati estratti e blocca il coordinatore che deve aspettare che tutte le posizioni siano state assegnate. Controlla che il buffer sia full, se lo è si alterna la stampa con l'ultimo processo

Processo Attivo :

- ① Attende estrazione
- ② cerca di accedere alla posizione del buffer
 - Vince
 - ③ Scrive il proprio PID
 - ④ Non partecipa più alla contesa
- ⑤ Se è l'ultimo stampa e legge dal buffer

Processo Attivo :

```
wait(estratto) // Aspetto l'estrazione
lock(my_mutex)
if (indice != 0)
    buffer[indice] = pid()
indice ++
processo ++ // incremento il numero di processi finiti
else
    continue // passa alla prossima iterazione
unlock(my_mutex)
if (processo == 2 * N) { // Se il numero di processi finiti è 2 * N
    buffer == true
    Signal(coordinatore)
    if (isLastProcess()) {
        for (ind_stampa = 0 to N) {
            wait(turno_ultimo_processo)
            stampo(buffer[ind_stampa])
            signal(turno_coordinatore)
        }
    }
}
```

In un ristorante più persone siedono allo stesso tavolo e condividono **B1 bottiglie d'acqua** e **B2 bottiglie di vino**.

La politica della mensa è che è un **operatore** della mensa controlla se **terminano le bottiglie** e le **sostituisce se sono vuote**, fino a quando le persone non decidono di **lasciare il ristorante**.

Operatore :

- ① Controlla se terminata bottiglia d'acqua
↳ ② Sostituisce la bottiglia d'acqua
- ③ Controlla se terminata la bottiglia di vino
↳ ④ Sostituisce la bottiglia di vino

Personne .

- ① Arrivano al ristorante
- ② Scelgono se bere acqua o vino
- ③ Bevono
- ④ Lasciano il ristorante

var :

Acq - Full : semaforo . = 1

Vino - Full : semaforo . = 1

dienti : semaforo : 1

B1 : array of [N]

ind - A : integer

Acq - empty : semaforo . = 0

Vino - empty : semaforo . =

my - mutex : MUTEX : = 1

B2 : array of [N]

ind - B : integer

Operatore :

repeat

```
lock ( my - mutex )
if ( dienti > 0 )
unlock ( my - mutex )
```

Forse si può fare
semplicemente una wait()

controllo_bottiglie ()

if (acq - empty = 1)

lock (my - mutex)

sostituisco_acqua ();

signal (acq - full)

unlock (my - mutex)

else if (vino - empty = 1)

lock (my - mutex)

sostituisco_vino ();

signal (vino - full)

unlock (my - mutex)

else

wait (operatore)

else

wait (operatore)

forever

Personne .

lock (my - mutex)

if (scelta - cliente = = " Acqua ")

unlock (my - mutex)

wait (acq - full)

lock (my - mutex)

riempie_bicchiere ()

B1 [ind - A --]

unlock (my - mutex)

signal (acq - empty)

lock (my - mutex)

else if (scelta - cliente = = " Vino ")

unlock (my - mutex)

wait (vino - full)

lock (my - mutex)

riempie_bicchiere ()

B2 [ind - v --]

unlock (my - mutex)

signal (vino - empty)

else

lascia_ristorante ()

In un sistema sono attivi N (N multiplo di 3) processi che si **autorganizzano** in **tre gruppi** seguendo il seguente schema:

- Inizialmente **tutti** competono per entrare nel gruppo **G1** ma solo $N/3$ vi accedono.
- I **restanti processi** competono per entrare nel gruppo **G2** ma solo $N/3$ vi accedono.
- I **restanti $N/3$ processi** sono assegnati al gruppo **G3**.

Creati i gruppi, i processi appartenenti al gruppo **G1**, a **turno ed all'infinito**, scelgono un **gruppo random G** ($G2$ o $G3$), un **numero random P** in $[0, N/3-1]$ e **producono** un elemento che verrà **consumato** dal processo **P** appartenente al gruppo **G**.

Processo :

- ① Tutti i processi vogliono entrare in G_1
 - ④.1 Entrano solo $N/3$
- ② I processi restanti vogliono entrare in G_2
 - ④.2 Entrano solo $N/3$
- ③ I restanti vengono assegnati a G_3
- ④ I processi di G_1 a turno e all'infinito
 - ④.3 Scelgono un gruppo casuale G
 - ④.4 Scelgono un numero casuale P
- ⑤ Producono un elemento
- ⑥ Che verrà consumato dal processo P del gruppo G

Var:

q_1 : array of $[N/3]$	$vuoto$: semaforo := 1	ind_q_1 : integer := 0	$gruppo$: var := {0,1,2}	$prodotto$: array of $[N]$
q_2 : array of $[N/3]$	$pieno$: semaforo := 0	ind_q_2 : integer := 0	q : integer := 0	$consumuto$: integer := 0
q_3 : array of $[N/3]$	my_mutex : mutex := 1	ind_q_3 : integer := 0	P : integer := 0	

Processo (1)

lock (my_mutex)

if ($ind_q_1 < N/3$) {

$q_1[ind_q_1] = processo$

ind_q_1++

$gruppo = 0$

}

else if ($ind_q_2 < N/3$) {

$q_2[ind_q_2] = processo$

ind_q_2++

$gruppo = 1$

}

else {

$q_3[ind_q_3] = processo$

ind_q_3++

$gruppo = 2$

unlock (my_mutex)

for ($i = 0$ to $N/3$) {

wait (vuoto)

lock (my_mutex)

if ($gruppo == 0$)

$q = rand(4,2)$

$P = rand(0, N/3-1)$

$prodotto = produco_elemento()$

unlock (my_mutex)

signal (pieno)

}

wait (pieno)

if ($prodotto != null$)

lock (my_mutex)

$consumuto = consuma(prodotto, P, q)$

unlock (my_mutex)

signal (vuoto)

- In una fabbrica, N operai preparano **piastrelle** da far cuocere in un forno, capace di cuocerne M contemporaneamente.
- All'uscita dal forno K operai visionano le piastrelle per **decorarle** secondo tale sequenza di passi:
 - Se trova una piastrella **difettata** inizia a prenderne dal forno **2 alla volta**;
 - Altrimenti ne prende **1 alla volta**;

Operario :

- ① Prepara M piastrelle
- ② **InForno**
- ③ Toglie le M piastrelle dal forno

M piast.



Decoratore :

- ① Aspetta che le piastrelle siano pronte per essere decorate
- ② Controlla se la piastrella è difettata
 - Si → **③ Ne prende 2 alla volta**
 - No → **④ Ne prende 1 alla volta**

Vari:

Forno - empty : semaforo := 1

Forno : array of [M]

decora : integer := 0

Forno - full : semaforo := 0

ind - Forno : integer := 0

my - mutex : Nutrex := 1

piastrella : boolean := False

Operario :

```

wait (forno - empty)
lock (my - mutex)
for (i=0 to N)
  Forno [ind - f] = inForno.piastrelle ()
  ind - f ++
unlock (my - mutex)
signal (forno - full)
  
```

L'operario attende che il forno sia vuoto così da infornare le piastrelle, dopodiché in M.E le inforna tutte ed N e quando ha finito segnala che il forno è pieno

Decoratore :

```

wait (forno - full)
lock (my - mutex)
for (i=0 to M) {
  piastrella = controllo (Forno [ind - Forno])
  if (piastrella == true) {
    for (i=0 to 2) {
      decoro = Forno [ind - Forno]
      ind - Forno --
    }
  } else {
    decoro = Forno [ind - Forno]
    ind - Forno --
  }
  unlock (my - mutex)
  signal (forno - empty)
}
  
```

Il decoratore quando il forno è pieno inizia a controllare le piastrelle se ne trova 1 difettata ne prende 2 alla volta finché non finiscono le piastrelle, altrimenti ne prende 1, dopodiché segnala che il forno è vuoto

In un sistema sono attivi **N** processi dotati di un **credito C**.

- Ogni processo **genera** un **numero casuale P** in $[1..C]$ e **verifica** se il proprio **P** è **maggiore** di quello degli altri processi (a parità di P vince il processo con PID minore).
- Il processo **vincitore** (quello con P maggiore) **aggiunge** al proprio credito **C** le **quantità P generate da tutti gli altri processi**, che invece **sottraggono** al proprio credito **C** la quantità **P** generata. Un processo con credito **C=0** termina.

Processo (i):

- ① Genera un numero casuale P in $[1..c]$
- ② Se il credito è 0 termina

Vari:

processo : semaforo := 1

num_generato : semaforo := 0

my_mutex : Mutex := 1

P : array of [N]

C : array of [N]

winner : integer := 0

Operazioni :

- ① Verifica se il valore P è > di quello degli altri processi
- ② Quello con P maggiore (vincitore) aggiunge al proprio credito le quantità P generate dagli altri processi
- ③ Se invece non sei il vincitore sottra al proprio credito la quantità P generata.

Processo (i):

repeat

wait (processo)

lock (my_mutex)

$P[i] = \text{genera_num}(\text{rand}(1..C))$

$C[i] = 0$

i++

unlock (my_mutex)

signal (num_generato)

forever

Operazioni :

repeat

wait (num_generato)

lock (my_mutex)

winner = controllo_P (P)

for (j=0 to N-1)

if (controllo_credito($P[j], C$) == 0)

(Termina il processo)

exit()

if (j == winner)

$C[winner] += P[i]$

else

$C[j] -= P[i]$

unlock (my_mutex)

signal (processo)

forever

I passeggeri in transito in un **aeroporto** prima di imbarcarsi sull'aereo, devono superare i **controlli di sicurezza**, i controlli sono gestiti come segue:

- Ogni **passeggero** sceglie tra **N code quella più breve** a cui accodarsi ed attendere il proprio turno per passare il controllo.
- Ogni coda è gestita da un **addetto** che lascia passare il primo passeggero della coda, **indirizzandolo** verso una delle **M code di accesso al metal detector**. Ogni addetto indirizza il passeggero verso la **coda più breve con lunghezza minore di K**.
- Se il **numero di code con lunghezza minore di K** è **inferiore a M/2**, gli addetti **NON** lasciano passare i passeggeri finchè persiste tale condizione. Dopo aver attraversato il metal detector, il passeggero si dirige all'imbarco.

Passeggero :

- ① Entra in aeroporto
- ② Sceglie la coda più breve
- ③ Passa il primo controllo
- ④ Attende indirizzamento dell'addetto
- ⑤ Attraversa Metal Detector e va all'imbarco

Var:

passeggero : semaforo := 0

addetto : semaforo := 1

scelta_effettuata . semaforo := 0

my_mutex : MUTEX := 1

Coda : array of [N]

MD : array of [M]

Addetto :

- ① Attende passeggero
- ② Controlla che il numero di code con lunghezza minore di K sia inferiore a M/2
 - Si → ②.1 Non lascia passare
- ③ Indirizza il passeggero verso la coda più breve

Passeggero :

repeat

 signal (passeggero)

 lock (my_mutex)

 mia_scelta = coda_min (Coda[i])

 mia_scelta ++

 signal (scelta_effettuata)

 unlock (my_mutex)

 wait (addetto)

 {attraversa_Metal_Detector}

forever

Addetto :

repeat

 wait (scelta_effettuata)

 lock (my_mutex)

 if (coda[n] < MD[M/2])

 { Attendi }

 else

 scelta_add = min (MD[ind_no])

 ind_md ++

 scelta_add ++

 unlock (my_mutex)

 signal (addetto)

forever

Per comodità ho stabilito che lo SLAVE che vince consuma tutto.

In un sistema sono attivi N processi slave ed un processo master.

- Ogni processo slave inserisce in un buffer condiviso di N posizioni un numero random M , con $M \leq N/2$. Quando tutti gli slave hanno inserito un numero si mettono in attesa, mentre il master, trovato il numero massimo (Max), produce Max elementi, e risveglia processi slave che competeranno per consumare gli elementi disponibili. Terminati gli elementi da consumare, i processi slave ed il master ripetono il procedimento dall'inizio.

Slave (N):

- Genera un numero casuale M (con $M \leq N/2$)
- Inserisce in un buffer condiviso il numero generato
- Attivo il master
- Si mettono in attesa e attendono di essere risvegliati
- Consumano gli elementi prodotti dal processo Master

Master :

- Attende i processi slave
- Trova il numero massimo (Max)
- Produce Max elementi e li mette in un nuovo Buffer
- Risveglia processi slave

Var:

master : semaforo := 0
risveglio : semaforo := 0
my_mutex : MUTEX := 1

Buffer : array of [N]
Elementi_disp : array of [N]

num_mass : integer := 0
M : integer = 0

Slave :

repeat
lock (my_mutex)
 $M = \text{rand}(1 \dots N/2)$
Buffer[ind] = M
ind++
unlock (my_mutex)
Signal (master)
lock (my_mutex)
svuotaBuffer (Buffer[ind])
unlock (my_mutex)
wait (risveglio)
lock (my_mutex)
for (j=N to 0)
consumo (Elementi_disp)
Elementi_disp[ind-disp]--
unlock (my_mutex)
forever

Genero un numero randomico < $M/2$ e lo inserisco nel buffer dopodiché attivo il master e svuoto il buffer e attendono il risveglio. Una volta risvegliato faccio un for inverso dove vedo a consumare dal buffer nuovo

Master :

wait (master)
lock (my_mutex)
num_mass = mux (Buffer[ind])
Elementi_disp [ind disp] = produci (num_mass)
ind disp++
unlock (my_mutex)
signal (risveglio)

Aspetto i processi slave entro in M.E , calcolo il massimo del buffer Slave e produco un nuovo buffer di max elementi .
Risveglio gli slave

Nella sala recruitment di una grande azienda informatica, durante una sessione di interview possono prendere posto al piu' N candidati, mentre i restanti candidati prenderanno parte alla sessione di interview successiva. Ogni candidato che ha trovato posto sara' chiamato al tavolo delle interview. L'addetto alle risorse umane attende fino che e' disponibile un candidato. Terminata l'intervista il candidato lascia il tavolo e la sala facendo posto al prossimo candidato. Le sessioni di interview terminano quando NON ci sono piu' candidati in attesa. Proporre una soluzione e discutere eventuali problemi di deadlock.

Candidati :

- ① Vedo se posso entrare nella sessione
 - ①.1 Oppure mettermi in attesa per la prossima sessione
- ② Aspetto di essere chiamato per l'intervista
- ③ Faccio l'intervista
- ④ Segnalo fine intervista
- ⑤ Lascio tavolo e sala

Addetto :

- ① Attendo candidato per l'intervista
- ② Chiama il candidato al tavolo
- ③ Fa intervista
- ④ Controlla se ci sono altri candidati
 - ↪ ④.4 Termina sessione

VAR :

```
candidati : semaforo -> 0
candidatoDisponibile : semaforo -> 0
tavolo : semaforo -> 0
inAttesa : integer -> 0
my-mutex : MUREX -> 1
```

Candidato :

```
lock(my-mutex)
if(postiOccupatiSala < N) {
    postiOccupatiSala ++
    unlock(my-mutex)
    wait(intervista)
    fui-intervista()
    wait(Fine-intervista)
    signal(tavolo)
    Esci()
    lock(my-mutex)
    postiOccupatiSala --
    unlock(my-mutex)
} else {
    lock(my-mutex)
    inAttesa ++
    unlock(my-mutex)
}
```

Addetto :

```
signal(intervista)
wait(tavolo)
fui-intervista()
signal(Fine-intervista)
lock(my-mutex)
if(postiOccupatiSala == 0)
    signal(nuova-sessione)
unlock(my-mutex)
```

DA CONTROLLARE

Il chitarrista Mancuso ha una masterclass in cui possono prendere parte **N** chitarristi, che possono scegliere fra **M modelli di chitarre**, con **K chitarre** per modello.

Quando tutti gli allievi hanno chitarra, inizia masterclass quelli **che non trovano posto**, prenderanno parte alla successiva tutto termina quando **NON ci sono chitarristi in attesa**.

Chitarristi :

- ① Controlla se ci sono posti:
↳ NO
 ①.1 Partecipano alla prossima lezione
- ② Scelgono tra M modelli di chitarra, dove ciascuno ha K chitarre

Vari:

```
inAttesa : integer := 0
postiDisponibili : integer := 0
chitarra_seelta : integer := 0
```

```
ind_M : integer := 0
ind_K : integer := 0
chitarre : array of [M][K]
```

```
scelta_semaforo := 0
mancuso : Semaforo := 1
finelezione : semaforo := 0
```

```
my_mutex : MUTEX := 1
```

Mancuso :

- ① Controllo che tutti gli allievi abbiano la chitarra
- ② Inizia lezione
- ③ Fine lezione
- ④ Se non ci sono allievi in attesa termina lezione

Chitarristi :

lock (my_mutex)

if (postiDisponibili > 0)

Controlla che ci siano posti disponibili

unlock (my_mutex)

dopo di che scelgo la chitarra e comunque

lock (my_mutex)

che ho scelto.

Scelgo chitarra

chitarra_seelta := scelti (chitarre [ind_M] [ind_K])

ind_K --

signal (scelta)

unlock (my_mutex)

wait (mancuso)

Attendo il mancuso per l'inizio e

wait (finelezione)

la fine della lezione, dopo di che

lock (my_mutex)

dopo la fine incremento il num. di

postiDisponibili (chitarra [ind_M] [ind_K])

chitarre e di posti Disponibili.

ind_K ++

postiDisponibili ++

{Lascio la lezione}

unlock (my_mutex)

else {

Se non ci sono posti aumenta

lock (my_mutex)

l'attesa per l'addetto.

inAttesa ++

{ non ci sono posti quindi attendo }

unlock (my_mutex)

```
int main() {
```

```
    crea - processo (mancuso)
```

```
    for (i=0 to N) {
```

```
        crea - processo (chitarrista)
```

```
}
```

Mancuso :

lock (my_mutex)

if (inAttesa > 0) { Se ci sono persone

unlock (my_mutex)

in attesa allora

for (i:=0 to N) {

attendo la scelta

wait (scelta);

e inizio la lezione

signal (mancuso)

dopo la lezione

inizialLezione()

segnalo la fine.

signal (finelezione)

else {

{ Termina sessione }

