Esercizi sulla sincronizzazione dei processi

- i semafori
- la mensa universitaria
- il distributore di benzina
- i lettori e gli scrittori
- La finale della coppa del mondo
- Il grande magazzino
- Il barbiere sonnolento

Semafori

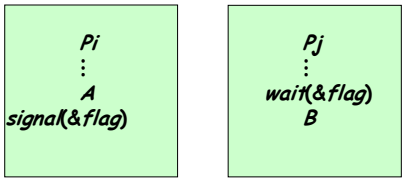
- I semafori sono strumenti di sincronizzazione
- Il semaforo contiene una variabile intera S che serve
  - per proteggere l'accesso alle sezioni critiche
  - Sincronizzare i processi
- Si può accedere al semaforo (alla variabile) solo attraverso due operazioni atomiche
  - ***wait(S):*** il processo chiede di accedere alla propria sezione critica. Esso aspetta se altri processi sono dentro la loro sezione critica
  - ***signal(S):*** il processo vuole uscire dalla propria sezione critica

definizione di wait e signal

wait (S)
- se S <= 0 il processo attende che S > 0
- Altrimenti (S > 0) allora S=S-1 e il processo continua l'esecuzione

signal(S)
- esegue S=S+1

Esempio: flag=0



B verra' eseguito sempre dopo di A

1: la mensa universitaria

- In una mensa universitaria, gli studenti, dopo aver mangiato, depongono i vassoi in M contenitori, ognuno di K ripiani. Periodicamente, un addetto alle cucine, sceglie 1 contenitore tra quelli in cui non ci sono piu' ripiani liberi , lo svuota, lava i piatti e riporta il contenitore in sala.
- Si descriva una soluzione in uno pseudo linguaggio che ottimizzi l'accesso alle risorse usando semafori e processi

soluzione (1/2)

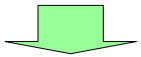
```
sem cont[M] = 1  // 1 semaforo per contenitore. Tutti inizializzati a 1
int liberi[M]  = K // array di interi condiviso. Tutti inizializzati a K

void cameriere( )
i = -1
while ( 1 )
   i = ( i+1 ) %M
   wait (cont[ i ])
         if (liberi [i] == 0)
                  liberi [ i ] = K
                  lava ( )
         endif
    signal (cont[ i ])
endwhile
end cameriere
```


soluzione (2/2)

```
void studente ( )
i = -1
fatto == no
while ( fatto == no )
     i = ( i+1 ) %M
    wait (cont[ i ] )
         if (liberi [i] > 0)
                  liberi [ i ] = liberi [ i ] - 1
                  posa ( )
                  fatto = si
         endif
    signal (cont[ i ] )
endwhile
end studente
```


problema

- la soluzione proposta e' tale che se uno studente sceglie il contenitore che l'addetto sta liberando (e lavando) aspetta (con il vassoio in mano) che l'addetto lo riporti in sala



fare in modo che lo studente torni subito in aula
ad ascoltare la lezione del professore

seconda soluzione (1/2)

```
sem cont[M] = 1  // 1 semaforo per contenitore. Tutti inizializzati a 1
int liberi[M]  = K // array di interi condiviso. Tutti inizializzati a K
char disponibile[M] =si // array di char condiviso. Tutti inizializzati a "si"

void cameriere( )
i = -1
while ( 1 )
   i = ( i+1 ) %M
   wait (cont[ i ])
         if (liberi [i] == 0)
                  disponibile [ i ] = "no"
                  signal (cont[ i ])
                  liberi [ i ] = K
                  lava ( )
                  disponibile [ i ] = "si"
         else
                  signal (cont[ i ])
         endif
endwhile
end cameriere
```

soluzione (2/2)

```
void studente ( )
i = -1
fatto == no
while ( fatto == no )
    i = ( i+1 ) %M
    wait (cont[ i ] )
          if (disponibile [ i ] == si)
                if (liberi [i] > 0)
                liberi [ i ] = liberi [ i ] - 1
                posa ( )
                fatto = si
                endif
          endif
    signal (cont[ i ] )
endwhile
```


2: il distributore di benzina

- Un distributore di benzina ha N pompe e 1 serbatoio della capacita' di M litri. Ogni automobile all'arrivo richiede una specifica quantita' di benzina. Il serbatoio e' rifornito da una autobotte che lo riempie fino alla capacita' massima e solo se nessuna automobile sta facendo rifornimento.
- le automobili possono fare benzina solo se c'e' una pompa libera, se la quantita' di benzina richiesta e' disponibile e se l'autobotte non sta riempiendo il serbatoio
- Si descriva una soluzione in un pseudo linguaggio che ottimizzi l'accesso alle risorse usando semafori e processi

soluzione (1/2)

```
sem distributore  // semaforo inizializzato a 1
int occupate = 0 // intero condiviso inizializzato a 0
sem serbatoio // semaforo inizializzato a 1
int disponibile = M // intero condiviso inizializzato a M

void autobotte ( )
wait (distributore)
if ( occupate > 0)
    signal (distributore)
else
    riempi ( )
    disponibile = M
    signal (distributore)
endif
end autobotte
```


soluzione (1/2)

```
void automobile ( )
wait ( distributore )
if ( occupate < N )
    occupate = occupate + 1
    signal ( distributore )
    richiesta = rand (1 .. 20)
    wait (serbatoio)
    if (richiesta > disponibile )
          signal (serbatoio)
    else
          disponibile = disponibile – richiesta
          signal (serbatoio)
          faibenzina( )
    endif
    wait(distributore)
    occupate = occupate -1
    signal(distributore)
else
    signal (distributore)
endif
end automobile
```

problema

- la soluzione proposta fa si che
  - la (N+1)-ma auto va via
  - l'autobotte va via se c'e' almeno un'auto

fare in modo che l'autobotte abbia la precedenza ed aspetti che il distributore sia libero

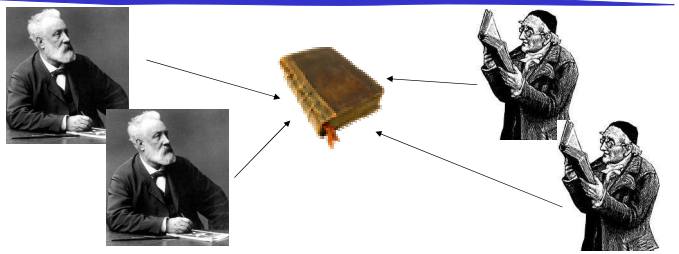
seconda soluzione (1/2)

```
sem distributore =1 // semaforo inizializzato a 1
int occupate = 0 // intero condiviso inizializzato a 0
sem serbatoio =1 // semaforo inizializzato a 1
int disponibile = M // intero condiviso inizializzato a M
sem  auto =1 // semaforo inizializzato a 1
char autobotte = no // char condiviso inizializzato a no

void autobotte ( )
wait (distributore)
autobotte = si
signal (distributore)
wait ( auto )
riempi ( )
disponibile = M
signal (auto)
autobotte = no
end autobotte
```


seconda soluzione (1/2)

```
void automobile ( )
wait ( distributore )
if ( occupate < N    and   autobotte == no)
     occupate = occupate + 1
    if (occupate == 1 ) wait (auto)
     signal ( distributore )
     richiesta = rand (1 .. 20)
     wait (serbatoio)
     if (richiesta > disponibile )
            signal (serbatoio)
     else
            disponibile = disponibile – richiesta
            signal (serbatoio)
            faibenzina( )
     endif
     wait( distributore )
    occupate = occupate-1
    if (occupate == 0) signal (auto)
     signal (distributore)
else
     signal (distributore)
endif
end automobile
```


3: i lettori e gli scrittori



- Un insieme di dati (ad es. un file) deve essere condiviso da più processi concorrenti che possono richiedere la sola lettura del contenuto, o anche un aggiornamento.
- Se due *lettori* accedono contemporaneamente ai dati condivisi non ha luogo alcun effetto negativo
- Se uno *scrittore* accede simultaneamente ad un altro processo si può avere incoerenza dell'informazione

2 tipi di problemi

- 1° problema dei lettori-scrittori: nessun processo lettore deve attendere, salvo che uno scrittore abbia già ottenuto l'accesso ai dati condivisi (precedenza ai lettori)

possibilità di starvation per gli scrittori

- 2° problema dei lettori-scrittori: nessun processo scrittore deve attendere, salvo che uno scrittore abbia già ottenuto l'accesso ai dati condivisi (precedenza agli scrittori)

possibilità di starvation per gli lettori.

soluzione al primo problema scrittori/lettori

- Variabili condivise:

```
semaphore mutex, wrt;
int readcounter;
// inizialmente mutex = 1, wrt = 1, readcounter = 0
```

Processo scrittore

```
wait(wrt);
   ...
   effettua la scrittura
   ...
signal(wrt);
```

Uso esclusivo ai dati condivisi

Altri possono accedere ai dati condivisi

soluzione al primo problema scrittori/lettori

Uso esclusivo a readcounter

Se sono il primo lettore fermo gli scrittori

Se sono l'ultimo lettore sblocco gli scrittori

Processo lettore

```
wait(mutex);
readcounter++;
if (readcounter == 1)  wait(wrt);
signal(mutex);
      ...
      si effettua la lettura
      ...
wait(mutex);
readcounter - -;
if (readcounter == 0)  signal(wrt);
signal(mutex);
```


4: la finale della coppa del mondo

Al termine della finale del campionato del mondo di calcio del 2006, vinta dalla nazionale italiana contro quella francese, la polizia tedesca ha il compito di dirigere i tifosi alle stazioni ferroviarie, evitando che gruppi di differenti nazioni vengano a contatto.

A tal fine, ogni volta che gruppi di tifosi devono attraversare una piazza, arrivando da strade diverse, il capo della polizia decide che:

- se la piazza e' libera puo' essere attraversata da qualunque tifoso
- se nella piazza c'e' almeno un tifoso italiano nessun tifoso francese puo' attraversare la piazza
- se nella piazza c'e' almeno un tifoso francese nessun tifoso italiano puo' attraversare la piazza

Si implementi una soluzione in termini di processi e semafori sviluppando in uno pseudo linguaggio le procedure per gli italiani e per i francesi

soluzione

```
variabili condivise
int   ita=0, fra=0
semafori binari
mutex1 = 1,  mutex2=1, piazza=1

void italiano ( ) {
    wait(mutex1)
          ita++
          if ( ita == 1) wait (piazza)
    signal(mutex1)

    attraversa( )

    wait(mutex1)
          ita - -
          if ( ita == 0 ) signal (piazza)
    signal(mutex1)
}
```


5: il grande magazzino (versione semplificata)

- Un edificio di N piani possiede un ascensore capace di accogliere un numero infinito di passeggeri, ognuno dei quali può richiedere di essere trasportato da un piano all'altro usufruendo dell'ascensore. Per penalizzare quanto meno possibile il servizio, l'ascensore tende a mantenere il proprio verso di movimento (verso l'alto o verso il basso), fino a giungere a fine corsa. I passeggeri si comportano come segue:
  - specificano un piano di partenza e uno di arrivo.
  - Aspettano l'ascensore non è presente al piano di partenza
  - scendono dall'ascensore quando quest'ultimo arriva al piano di arrivo.

Si implementi una soluzione in termini di processi e semafori sviluppando in uno pseudo linguaggio le procedure per i passeggeri e per l'ascensore.

Per semplicita':

- si usi la funzione rand(0.,N) per specificare il piano di partenza e quello di arrivo e si supponga che i valori ottenuti non siano mai uguali.
- l'ascensore va sempre dal piano 0 al piano N e viceversa, anche se non ci sono clienti in attesa (politica SCAN detta anche dell'ascensore)

Soluzione passeggero

```
semin(N+1),  semout(N+1)  ,  asc          semafori bin tutti =0
contain(N+1),  contaout(N+1)               contatori

void passeggero( )
partenza=rand(0.. N)
arrivo=rand(0..N)

    wait(m1)
          contain(partena)++                    accoda al piano
    signal(m1)
    wait( semin(partenza) )                     attendi ascensore
    entra( )
    wait(m1)
          contasu(partenza) - -                 riduci la coda
          if ( contasu ( partenza) == 0) signal (asc)   se ultimo sblocca ascensore
    signal(m1)

    wait(m3)
          contauscita ( arrivo) ++              prenota discesa
    signal(m3)
    Viaggia( )
    wait( semuscita (arrivo) )          attendi arrivo per uscire e terminare
```


Soluzione ascensore 1/2

```
void ascensore ( )
piano = -1;    dir = su
while (1) {
    if (dir ==su)  piano ++                 definisci piano
    else           piano --

    wait(m1)
    if (contain(piano) >0){                                 se c'e' qualcuno al piano che sale…
          for (  i =1 to contain(piano) )   signal(semin(piano) )   … sbloccali
          signal(m1)
          wait(asc )                                attendi che tutti entrano
    else
          signal(m1 )
    endif

    wait(m3)
    if(contauscita(piano)>0 ){                      se qualcuno deve uscire…
          for i = 1 to contauscita(piano)
                signal ( semuscita (piano) )              … sbloccali
          endfor
    }
    contauscita(piano) =0                           svuota lista uscita
    signal(m3)

    if(piano==N) dir =giu                           inverti direzione
    if(piano=0) dir =su
}
```

5: il grande magazzino (versione completa)

- Ripetere l'esercizio precedente con il vincolo ulteriore:
  - I passeggeri attendono se l'ascensore non è presente al piano di partenza o non va nel verso corretto.

Soluzione passeggero 1/2

```
semgiu(N+1), semsu(N+1), semuscita(N+1) , asc      semafori bin tutti =0
contagiu(N+1), contasu(N+1), contauscita(N+1)      contatori

void passeggero( )
partenza=rand(0.. N)
arrivo=rand(0..N)

if(partenza < arrivo) {                            passeggero deve salire

    wait(m1)
        contasu(partena)++                         accoda al piano
    signal(m1)
    wait( semsu(partenza) )                        attendi ascensore
    entra( )
    wait(m1)
        contasu(partenza) - -                      riduci la coda
        if ( contasu ( partenza) == 0) signal (asc)  se ultimo sblocca ascensore
    signal(m1)

} else {                                           passeggero deve scendere
```


Soluzione passeggero 2/2

```
} else {                                           passeggero deve scendere

    wait(m2)
        contagiu(partenza)++
    signal(m2)
    wait( semgiu(partenza) )
   entra( )
    wait(m2)
        contagiu(partenza)- -
        if ( contagiu ( partenza) == 0) signal (asc)
    signal(m2)
}

wait(m3)
    contauscita ( discesa) ++                      prenota discesa
signal(m3)
Viaggia( )
wait( semuscita (discesa) )                        attendi arrivo per uscire e terminare
```

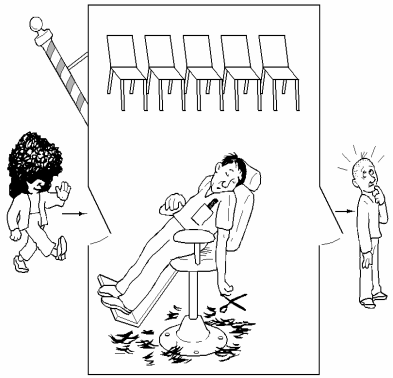

Soluzione ascensore 1/2

```
void ascensore ( )
piano = -1;   dir = su
while (1) {
    if (dir ==su)  piano ++                  definisci piano
    else           piano --

    wait(m1)
    if (contasu(piano) >0  && dir==su){          se c'e' qualcuno al piano che sale…
        for (  i =1 to contasu(piano) )    signal(semsu(piano) )     … sbloccali
        signal(m1)
        wait(asc )                                          attendi che tutti entrano
    else
        signal(m1 )
    endif

    wait(m2)
    if (contagiu(piano) >0  && dir=giu){
        for ( i =1 to contagiu(piano) ) signal(semgiu(piano) )
        signal(m2)
        wait(asc )
    else
        signal(m2 )                                                    continua
    endif
```

Soluzione ascensore 2/2

```
    wait(m3)
    if(contauscita(piano)>0 ){                  se qualcuno deve uscire…
          for i = 1 to contauscita(piano)
                signal ( semuscita (piano) )          … sbloccali
          endfor
    }
    contauscita(piano) =0                       svuota lista uscita
    signal(m3)

    if(piano==N) dir =giu                       inverti direzione
    if(piano=0) dir =su
}
```


6: il barbiere sonnolento



in un negozio di barbiere ci sono
- una sedia per lavorare
- n sedie per far attendere i clienti

- il barbiere di solito dorme
- quando arriva un cliente, questi lo sveglia e si fa servire
- se nel frattempo arrivano altri clienti, si accomodano sulle sedie oppure vanno via se sono tutte occupate

soluzione

```
#define CHAIRS 5                    /* # chairs for waiting customers */

typedef int semaphore;              /* use your imagination */

semaphore customers = 0;            /* # of customers waiting for service */
semaphore barbers = 0;              /* # of barbers waiting for customers */
semaphore mutex = 1;                /* for mutual exclusion */
int waiting = 0;                    /* customers are waiting (not being cut) */

void barber(void)
{
     while (TRUE) {
          wait  (&customers);       /* go to sleep if # of customers is 0 */
          wait  (&mutex);           /* acquire access to 'waiting' */
          waiting = waiting – 1;    /* decrement count of waiting customers */
       signal (&barbers);           /* one barber is now ready to cut hair */
       signal (&mutex);             /* release 'waiting' */
            cut_hair( );            /* cut hair (outside critical region) */
     }
}


void customer(void)
{
      wait  (&mutex);               /* enter critical region */
     if (waiting < CHAIRS) {        /* if there are no free chairs, leave */
          waiting = waiting + 1;    /* increment count of waiting customers */
       signal (&customers);         /* wake up barber if necessary */
       signal (&mutex);             /* release access to 'waiting' */
          wait  (&barbers);         /* go to sleep if # of free barbers is 0 */
        get_haircut( );             /* be seated and be serviced */
     } else {
       signal (&mutex);             /* shop is full; do not wait */
     }
}
```