Cenni ai Problemi classici di Sincronizzazione Processi Concorrenti

E. Mumolo

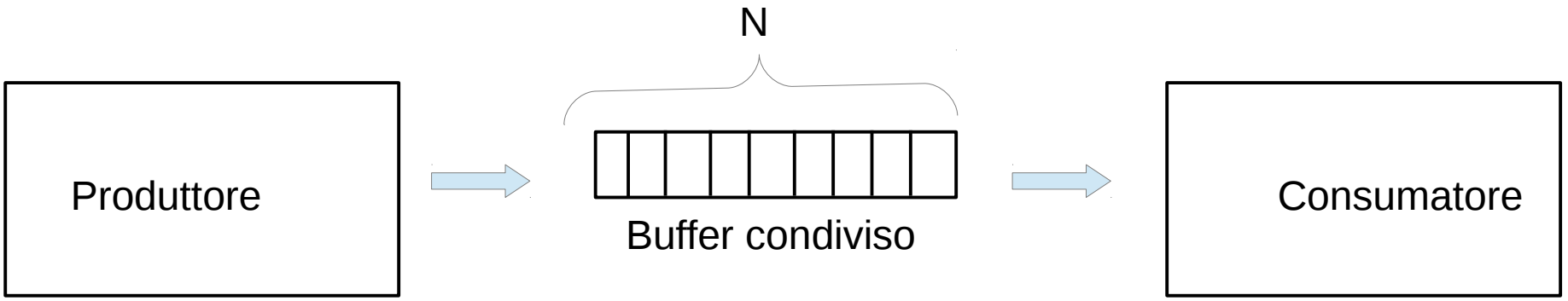
Problemi classici

- Problemi di mutua esclusione proposti nel corso degli anni
- Modelli di problemi reali
  - Problema del produttore consumatore
  - Problema dei Lettori/Scrittori
  - Problema dei 5 filosofi
  - Problema del ponte stretto
  - Problema del negozio del barbiere
  - Problema dei fumatori
  - Problema del parcheggio
  - ...

Il problema del Produttore/consumatore o *problema del buffer limitato* o *bounded-buffer problem*

- Il produttore e il consumatore condividono un buffer di dimensioni limitate
- Il produttore sospende la sua esecuzione se il buffer è pieno
- Il consumatore sospende la sua esecuzione se il buffer è vuoto
- Non appena il buffer ha spazio, il produttore vi deposita un elemento
- Non appena viene depositato un elemento, il consumatore lo preleva
- Ogni elemento viene prelevato una sola volta;
- Gli elementi sono prelevati nello stesso ordine in cui sono stati depositati
- La soluzione non deve in nessun caso andare in stallo
- Devono essere possibili più produttori o pi consumatori

Una soluzione semaforica al problema produttore-consumatore



Schema:
3 semafori: 1 binario, mutex
1 contatore, empty inizializzato a N
1 contatore, full inizializzato a 0

Produttore:

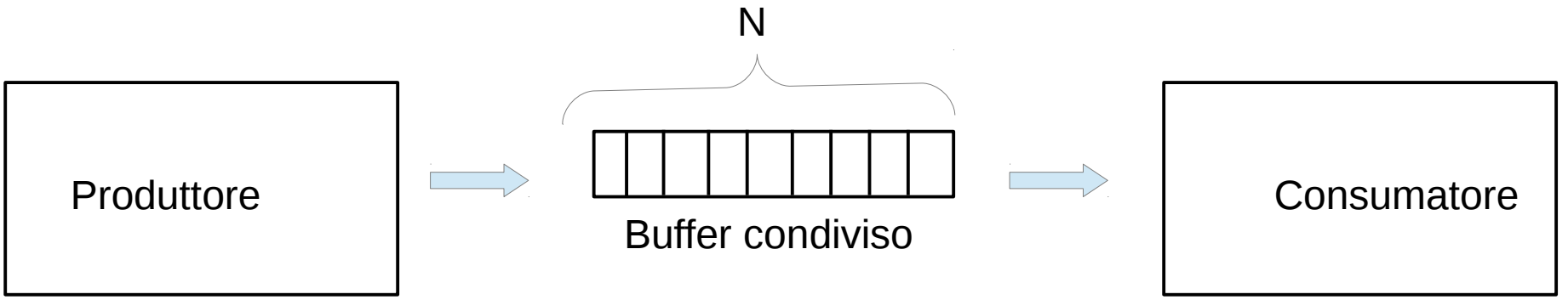
```
el=produci();

down(empty) ;
down(mutex);
Inserisci elp;
up(mutex);
up(full)    ;
```

Consumatore:

```
down(full);
down(mutex);
estrai elc;
up(mutex);
up(empty);
consuma(elc);
```

Una soluzione semaforica al problema produttore-consumatore



Schema:
3 semafori: 1 binario, mutex
1 contatore, empty inizializzato a N
1 contatore, full inizializzato a 0

Produttore:

```
el=produci();
down(mutex);
down(empty) ;
Inserisci el;
up(full)    ;
up(mutex);
```

ATTENZIONE:
Questa soluzione porta allo stallo del prod e del cons

Consumatore:

```
down(mutex);
down(full);
estrai el;
up(empty);
up(mutex);
consuma(el);
```

Produttore

```
int main (void)
{
     int fd, i, pid, size=(BUF_SIZE+2)*sizeof(float);
     float* buffer;
     sem_t* mutex;
     sem_t* vuoto;
     sem_t* pieno;
     int* in;
     int* out;
     float value, elto=0;

     mutex=sem_open("/mutex", O_CREAT, S_IRUSR | S_IWUSR, 1);    /* creo i semafori */
     vuoto=sem_open("/vuoto", O_CREAT, S_IRUSR | S_IWUSR, BUF_SIZE);
     pieno=sem_open("/pieno", O_CREAT, S_IRUSR | S_IWUSR, 0);

     shm_unlink("mymem");
     fd = shm_open("mymem",  O_CREAT | O_EXCL |O_RDWR, S_IRUSR | S_IWUSR);/* creo la shmem */
     ftruncate(fd,size);
     buffer = mmap(NULL ,size,PROT_READ | PROT_WRITE, MAP_SHARED ,fd, 0);

     in = (int*) (buffer + BUF_SIZE*sizeof(float));
     out = (int*) (buffer + (BUF_SIZE+1)*sizeof(float));

     *in = *out = 0;

     for(i=0;i<100;i++) {
          sem_wait(vuoto);
          sem_wait(mutex);
               usleep(500000); /*aspetta 1/2 secondo */
               value=elto++;
               buffer[*in] = value;  /* scrivo il buffer */
               *in = (*in + 1) % BUF_SIZE;
          sem_post(mutex);
          sem_post(pieno);
          printf ("ho scritto  %f  \n", value);
     }
     shm_unlink("mymem");
}
```

Consumatore

```
int main (void)
{
     int fd, i, pid, size=(BUF_SIZE+2)*sizeof(float);
     float* buffer;
     sem_t* mutex;
     sem_t* vuoto;
     sem_t* pieno;
     int* out;
     float value;

     mutex=sem_open("/mutex", O_CREAT, S_IRUSR | S_IWUSR, 1);    /* creo i semafori */
     vuoto=sem_open("/vuoto", O_CREAT, S_IRUSR | S_IWUSR, BUF_SIZE);
     pieno=sem_open("/pieno", O_CREAT, S_IRUSR | S_IWUSR, 0);

     fd = shm_open("mymem",  O_EXCL |O_RDWR, S_IRUSR | S_IWUSR); /* mi collego alla shmem */
     buffer = mmap(NULL ,size,PROT_READ | PROT_WRITE, MAP_SHARED ,fd, 0);

     out = (int*) (buffer + (BUF_SIZE+1)*sizeof(float));

     while(1) {
          sem_wait(pieno);
          sem_wait(mutex);

          value = buffer[*out]; /* leggo il buffer */
          *out = (*out + 1) % BUF_SIZE;

          sem_post(mutex);
          sem_post(vuoto);
          printf ("\tho letto %f\n", value);
     }
}
```

Attenzione: ai due codici precedenti bisogna aggiungere questi #include e #define:

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/mman.h>
#include <sys/stat.h>        /* mnemonici in mode */
#include <fcntl.h>           /* mnemonici in oflag */
#include <semaphore.h>

#define BUF_SIZE 5
```

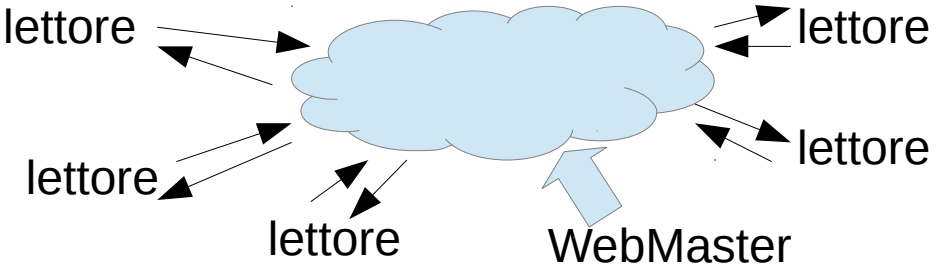
...che non sono stati inseriti per mancanza di spazio

Il problema dei Lettori/scrittori

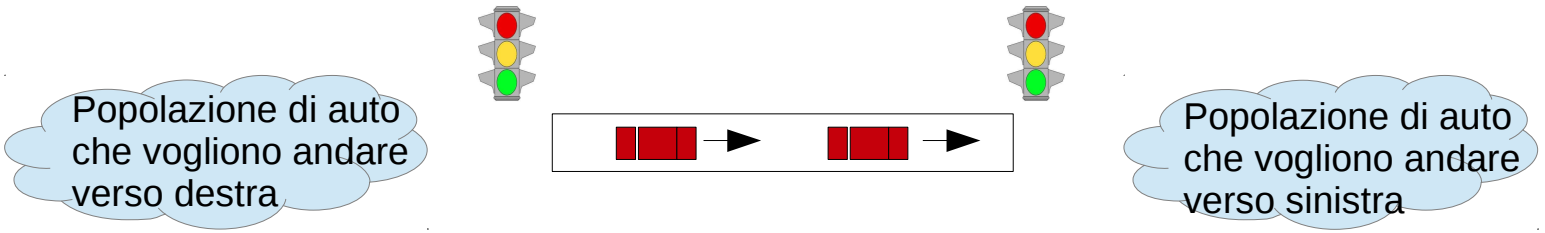
- Ci sono N thread che leggono una risorsa senza modificarla
- Ci sono M thread che modificano la risorsa
- Ci possono essere 1...Max numero di lettori concorrenti
- Lettori in mutua esclusione con scrittori
- Scrittori in mutua esclusione con tutti
- Gli scrittori hanno priorità sui lettori
- La soluzione non deve avere scrittori che aspettano un tempo indefinito (starvation)

Il problema dei Lettori/scrittori

- Può essere usato per gestire in concorrenza un archivio
- Può essere usato per gestire un sito web:



- Può essere usato per gestire una strada a senso unico:



In questo caso gli scrittori sono le auto in senso opposto → Problema del Ponte Stretto

Il problema dei Lettori/scrittori

- Accezione: lettura di un sito Web
- Principio di funzionamento:
  - Lo scrittore scrive in mutua esclusione protetto dal semaforo db
  - Il lettore:
    - Incrementa il numero di lettori concorrenti
    - Solo il primo lettore, fa una wait(db) (semaforo rosso)
    - I lettori leggono il data base in concorrenza
    - Una volta letto l'archivio, prima di terminare, il lettore decrementa il numero di lettori concorrenti

Scrittore

```
void main (int argc, char **argv)
{
     int fd, i, j, pid, size=(BUF_SIZE+1)*sizeof(float);
     float* buffer;
     sem_t* mutex;
     sem_t* db;
     int* nr_lett;
     float value, elto=0;
     time_t t;

     srand((unsigned) time(&t));

     mutex=sem_open("mutex", O_CREAT, S_IRUSR | S_IWUSR, 1); /* creo i semafori */
     db=sem_open("db", O_CREAT, S_IRUSR | S_IWUSR, 1);

     fd = shm_open("mymem",  O_CREAT | O_EXCL |O_RDWR, S_IRUSR | S_IWUSR);/* creo la SHM */
     ftruncate(fd,size);
     buffer = mmap(NULL ,size,PROT_READ | PROT_WRITE, MAP_SHARED ,fd, 0);

     nr_lett=(int*)(buffer+BUF_SIZE);

     *nr_lett=0;
     for(i=0;i<BUF_SIZE;i++) buffer[i]=(float)(rand() % 50);/*scrivo il database*/
     printf("scritto archivio\n");

     while(1){
        sleep(6);
        sem_wait(db);
        for(i=0;i<BUF_SIZE;i++) {buffer[i]=(float)(rand() % 50);printf("db %2.1f",buffer[i]);}
        printf(" Ho scritto l'archivio\n");
        sem_post(db);
     }
}
```

Lettore

```
int main (void){
     int fd, i, pid, size=(BUF_SIZE+1)*sizeof(float);
     int* nr_lett;
     float* buffer;
     sem_t* mutex;
     sem_t* db;
     float value[10];

     mutex=sem_open("mutex",  O_EXCL, S_IRUSR | S_IWUSR, 1); /* attacco i semafori */
     db=sem_open("db", O_EXCL, S_IRUSR | S_IWUSR, 1);

     fd = shm_open("mymem",  O_EXCL |O_RDWR, S_IRUSR | S_IWUSR); /* attacco la SHM */
     ftruncate(fd,size);
     buffer = mmap(NULL ,size,PROT_READ | PROT_WRITE, MAP_SHARED ,fd, 0);
     nr_lett = (int*)(buffer+BUF_SIZE);

     sem_wait(mutex);/*entro nella sezione critica*/
          printf("sono il lettore %d\n",*nr_lett);
          (*nr_lett)++;
          if(*nr_lett == 1) sem_wait(db);
     sem_post(mutex);/*esco dalla sezione critica*/

     for(i=0;i<BUF_SIZE;i++)value[i] = buffer[i]; /* leggo il buffer */

     sleep(1);

     sem_wait(mutex);/*entro nella sezione critica*/
          (*nr_lett)--;
          if(*nr_lett == 0) sem_post(db);
     sem_post(mutex);/*esco dalla sezione critica*/

     printf("lettura database:\n");
     for(i=0;i<BUF_SIZE;i++)printf(" %2.1f ",value[i]);
     printf("\n");
}
```

Header file

- I programmi devono usare I seguenti header file e define:

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/mman.h>
#include <sys/stat.h>        /* mnemonici in mode */
#include <fcntl.h>           /* mnemonici in oflag */
#include <semaphore.h>

#define BUF_SIZE 5
```

Il problema del negozio del barbiere

- Il barbiere ha una sala d'attesa con N sedie
- Se non ci sono sedie libere, il cliente se ne va
- Se ci sono sedie libere il cliente aspetta di essere servito
- Se non ci sono clienti il barbiere dorme
- Se il barbiere sta dormendo, allora il cliente lo sveglia e viene servito.

Il problema del barbiere

- Schema utilizzabile per descrivere un un qualsiasi locale che offre servizi
- Schema di massima:
  - 3 semafori: barbiere=0, cliente=0, mutex=1

```
Barbiere

while(1){
   down(cliente)
   up(barbiere)
   <Esegue il lavoro>
   down(mutex)
   numero_clienti--
   up(mutex)
}
```

```
 Cliente

up(cliente)
down(mutex)
numero_clienti ++
up(mutex)
down(barbiere)
<aspetta la fine del lavoro>
exit
```

Il problema del barbiere

- Gestione del numero limitato di sedie = N

```
Barbiere

down(cliente)
up(barbiere)
<Esegue il lavoro>
down(mutex)
numero_clienti--

up(mutex)
```

```
Cliente

down(mutex)
numero_clienti ++
if(numero_clienti>N){
      up(mutex)
      exit
}
up(mutex)
up(cliente)
down(barbiere)
<aspetta la fine del lavoro>
```

Il barbiere

```
void main ()
{
    int fd, i, j, pid, size=sizeof(int);
    int* buffer;
    sem_t* mutex;
    sem_t* barbiere;
    sem_t* cliente;
    int* nr_clienti;

    mutex=sem_open("mutex", O_CREAT, S_IRUSR | S_IWUSR, 1); /* creo i semafori */
    barbiere=sem_open("barb", O_CREAT, S_IRUSR | S_IWUSR, 0);
    cliente=sem_open("clien", O_CREAT, S_IRUSR | S_IWUSR, 0);

    fd = shm_open("mymem1",  O_CREAT | O_EXCL |O_RDWR, S_IRUSR | S_IWUSR); /* creo la
SHN */
    ftruncate(fd,size);
    buffer = mmap(NULL ,size,PROT_READ | PROT_WRITE, MAP_SHARED ,fd, 0);

    nr_clienti=buffer;
    //*nr_clienti=0;

    while(1){
        sem_wait(cliente);
        sem_post(barbiere);
        sleep(1);
        printf("barbiere: ho servito il cliente %d\n",*nr_clienti);
        sem_wait(mutex);
        *nr_clienti=*nr_clienti-1;
        sem_post(mutex);
    }
}
```

Il cliente

```
int main (void)
{
    int fd, i, pid, size=sizeof(int);
    int* buffer;
    sem_t* mutex;
    sem_t* barbiere;
    sem_t* cliente;
    int* nr_clienti;

    mutex=sem_open("mutex",  O_EXCL, S_IRUSR | S_IWUSR, 1); /* attacco i semafori */
    barbiere=sem_open("barb",  O_EXCL, S_IRUSR | S_IWUSR, 1);
    cliente=sem_open("clien",  O_EXCL, S_IRUSR | S_IWUSR, 0);

    fd = shm_open("mymem1",  O_EXCL |O_RDWR, S_IRUSR | S_IWUSR);    /* attacco la SHM */
    ftruncate(fd,size);
    buffer = mmap(NULL ,size,PROT_READ | PROT_WRITE, MAP_SHARED ,fd, 0);

    nr_clienti=buffer;

    printf("cliente: entro dal barbiere\n");
    sem_post(cliente);
    sem_wait(mutex);
    *nr_clienti=*nr_clienti+1;
    sem_post(mutex);
    sem_wait(barbiere);
    sleep(1);
    printf("cliente %d: esco dal barbiere\n", *nr_clienti);
}
```

Il problema del ponte stretto

- Il ponte è una strada ad unica corsìa
- Più macchine possono andare a esclusivamente a destra oppure a sinistra
- C'è un numero massimo di macchine
- Se c'e' almeno un auto in un verso, non ci possono essere macchine nel verso opposto
- Le auto non devono andare in starvation nè in stallo

Il problema dei fumatori di sigarette

- C'è un tabaccaio e tre fumatori
  - fumatore 1 possiede il tabacco
  - fumatore 2 possiede la carta
  - fumatore 3 possiede i fiammiferi

1) Il tabaccaio mette a disposizione 2 componenti a caso e si sospende
2) Il fumatore che possiede il terzo componente prepara la sigaretta e la fuma
3) Quando finisce, sveglia il tabaccaio e va a 1)

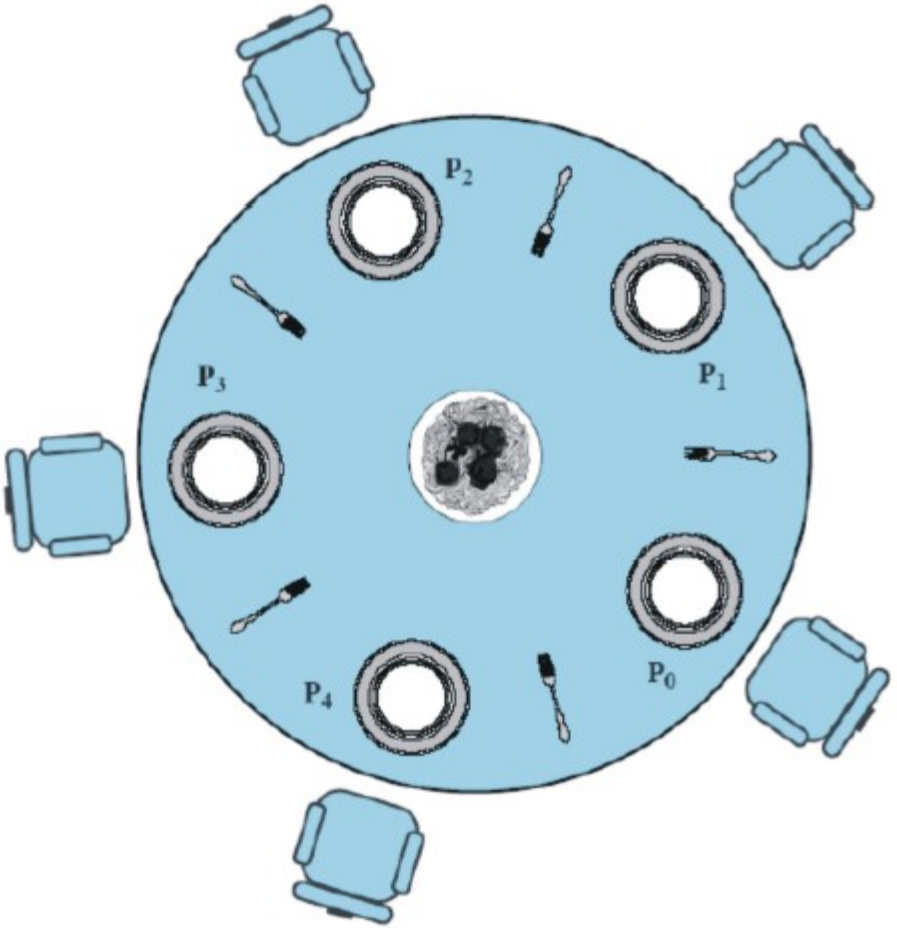
Il problema del parcheggio

- Un parcheggio a pagamento ospita tre tipi di veicoli
- Il parcheggio è diviso in spazi uguali
- I veicoli hanno dimensione diversa (esempio: auto, camper, autobus)
- I tre tipi di veicolo richiedono 1, 2 e 3 spazi
- Ogni veicolo viene gestito con un thread concorrente
- Il parcheggiatore viene gestito con un thread
- Scrivere il programma concorrente evitando la starvation

Il problema dei 5 filosofi

- Ci sono 5 filosofi intorno ad un tavolo
- I filosofi hanno il seguente ciclo vitale:
  - dormono-pensano-hanno appetito-mangiano
- Ciascun filosofo ha una forchetta a destra e una a sinistra
- Per mangiare, i filosofi devono prendere entrambe le forchette
- La soluzione deve permettere di mangiare al massimo numero di filosofi contemporaneamente
- La soluzione non deve andare nè in stallo nè in starvation

Il problema dei 5 filosofi



Il problema dei 5 filosofi

- Ogni filosofo è un thread concorrente
- Schema di massima:

```
filosofo(i)
{
	while(true){
		Pensa;
		Prende forchetta I
		Prende forchetta (i+1)%5
		Mangia;
		Deposita forchetta I
		Deposta forchetta (i+1)%5
	}
}
```

Il problema dei 5 filosofi

- Soluzione semaforica

```
Semaforo fork  = new Semaforo[5]
filosofo(i){
        while(true){
                Pensa;
                Prende forchetta i
                Prende forchetta (i+1)%5
                Mangia;
                Deposita forchetta i
                Deposta forchetta (i+1)%5
        }
}
```

- Il problema: se c'è un context switch tra prende forchetta sx e prende forchetta dx → Stallo!