La cena dei filosofi

Arturo Carpi

Dipartimento di Matematica e Informatica
Università di Perugia

Corso di Sistemi Operativi - a.a. 2020/21

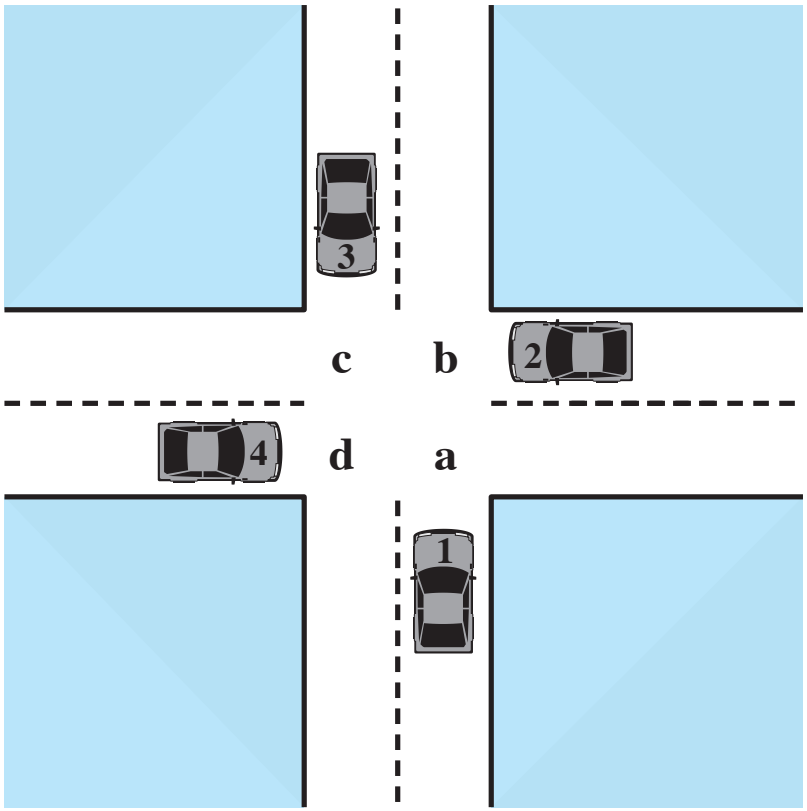
Stallo

Definizione

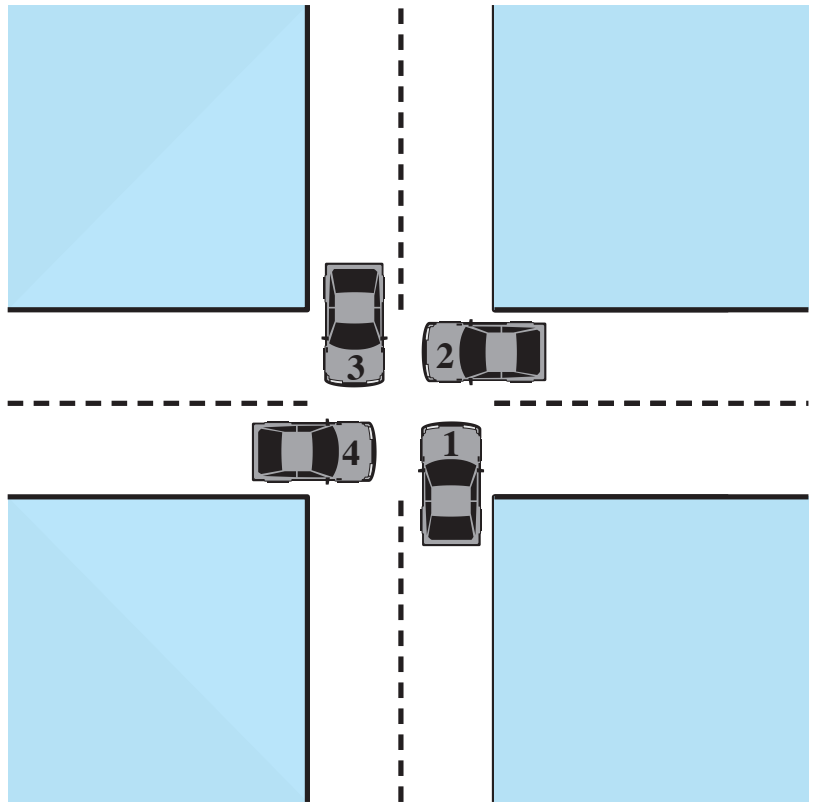
Un insieme di processi è in stallo (deadlock) se ogni processo è in attesa di un evento che può essere generato solo da un altro processo dell'insieme

permanente

nel caso generale, nessuna soluzione efficiente



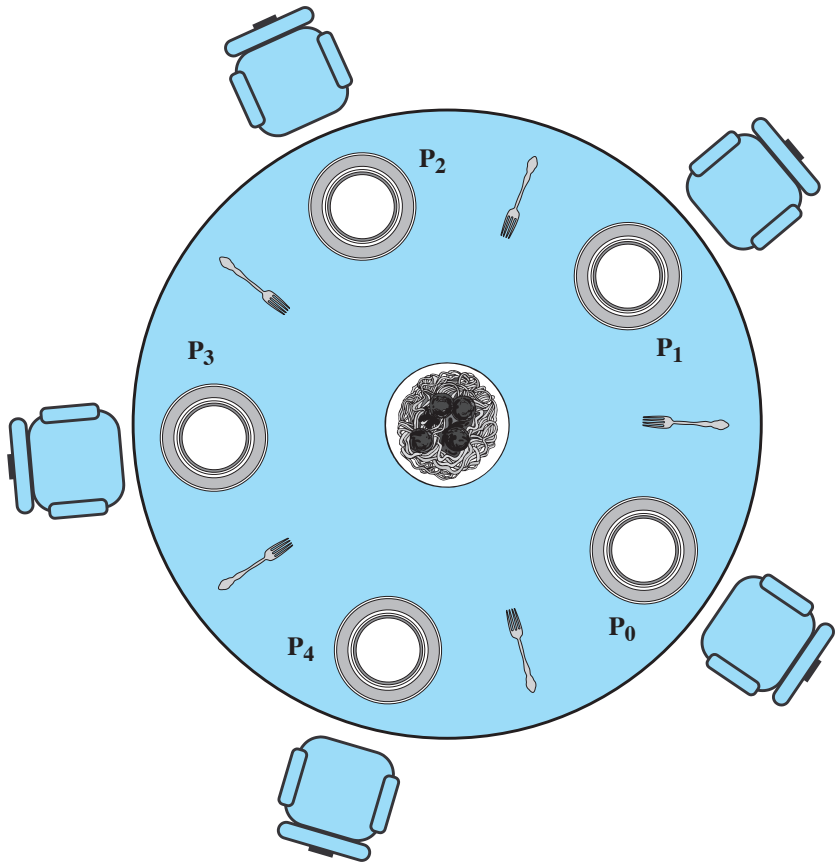
(a) Deadlock possible



(b) Deadlock

La cena dei filosofi

- Due filosofi non possono usare la stessa forchetta (mutua esclusione)
- Né stallo né starvation



Stallo possibile

```
/* program diningphilosophers */
semaphore fork [5] = {1};
int i;
void philosopher (int i)
{
   while (true) {
      think();
      wait (fork[i]);
      wait (fork [(i+1) mod 5]);
      eat();
      signal(fork [(i+1) mod 5]);
      signal(fork[i]);
   }
}
void main()
{
   parbegin (philosopher (0), philosopher (1), philosopher (2),
             philosopher (3), philosopher (4));
}
```

Stallo impossibile

```
/* program diningphilosophers */
semaphore fork[5] = {1};
semaphore room = {4};
int i;
void philosopher (int i)
{
   while (true) {
      think();
      wait (room);
      wait (fork[i]);
      wait (fork [(i+1) mod 5]);
      eat();
      signal (fork [(i+1) mod 5]);
      signal (fork[i]);
      signal (room);
   }
}
void main()
{
   parbegin (philosopher (0), philosopher (1), philosopher (2),
   philosopher (3), philosopher (4));
}
```

Soluzione con monitor

```
void philosopher[k=0 to 4]       /* the five philosopher clients */
{
  while (true) {
    <think>;
    get_forks(k);       /* client requests two forks via monitor */
    <eat spaghetti>;
    release_forks(k);   /* client releases forks via the monitor */
  }
}
```

```
monitor dining_controller;
cond ForkReady[5];              /* condition variable for synchronization */
boolean fork[5] = {true};       /* availability status of each fork */

void get_forks(int pid)         /* pid is the philosopher id number */
{
  int left = pid;
  int right = (++pid) % 5;
  /*grant the left fork*/
  if (!fork[left])
    cwait(ForkReady[left]);      /* queue on condition variable */
  fork[left] = false;
  /*grant the right fork*/
  if (!fork[right])
    cwait(ForkReady[right]);     /* queue on condition variable */
  fork[right] = false:
}
void release_forks(int pid)
{
  int left = pid;
  int right = (++pid) % 5;
  /*release the left fork*/
  if (empty(ForkReady[left])     /*no one is waiting for this fork */
    fork[left] = true;
  else                           /* awaken a process waiting on this fork */
    csignal(ForkReady[left]);
    /*release the right fork*/
    if (empty(ForkReady[right])  /* no one is waiting for this fork */
    fork[right] = true;
  else                           /* awaken a process waiting on this fork */
    csignal(ForkReady[right]);
}
```