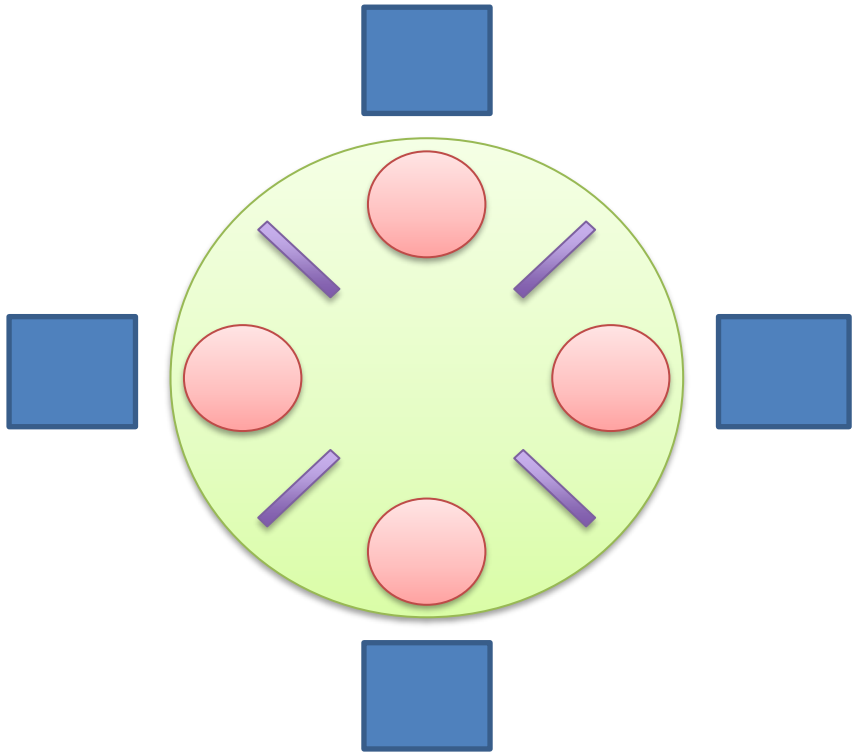
Filosofi a cena

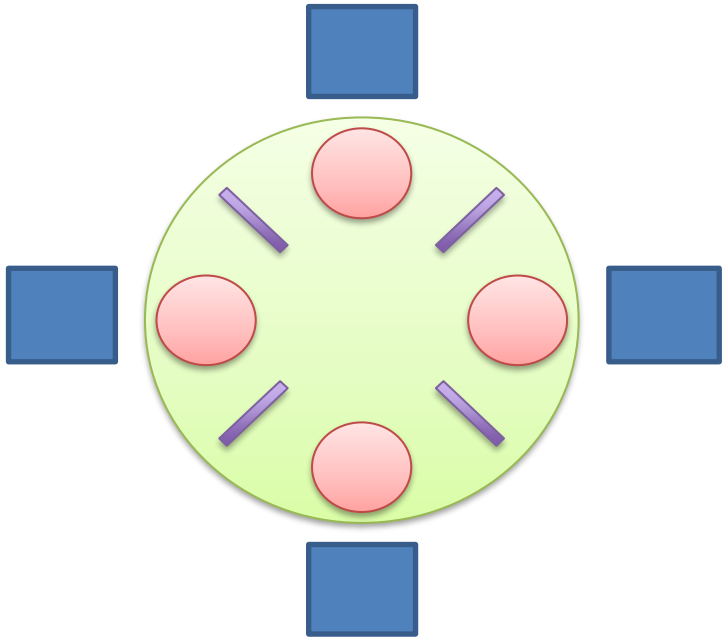
Il problema

N filosofi si ritrovano a cena in un ristorante cinese, occupando un tavolo circolare, apparecchiato con un piatto per ogni filosofo e un bastoncino interposto fra ogni coppia di piatti adiacenti.



Il problema

Per mangiare, ogni filosofo deve avere a disposizione *i due bastoncini* che si trovano rispettivamente alla sua sinistra e alla sua destra, entrando così in competizione con i due filosofi che siedono ai suoi lati. Il filosofo che non riesca ad ottenere ambedue i bastoncini (perché almeno uno è in possesso di un suo vicino) attende in attesa di ottenerli.



Il problema

Ogni filosofo alterna periodi in cui medita a periodi in cui vuole mangiare: quando decide di mangiare richiede i bastoncini ed eventualmente attende; dopo aver mangiato torna a pensare rilasciando entrambi i bastoncini.

Il problema può essere risolto associando una variabile di lock ad ogni bastoncino e organizzando queste variabili nel vettore bastoncino[i].

Il programma eseguito dal generico filosofo di indice i (i= 0, … ,N-1) è il seguente:

Soluzione 1

Filosofo i

```
…
while (true) {
    penso();   // il filosofo pensa
    // il filosofo di indice i decide di mangiare
    lockBastoncino[i].Acquire();   // acquisisce il bastoncino di destra
    // il filosofo si sospende se non può acquisire il bastoncino alla sua sinistra
    lockBastoncino[(i+ 1) mod N].Acquire();
    mangia();  // il filosofo di indice i mangia
    // rilascia i bastoncini
    lockBastoncino[(i+ 1) mod N].Release();
    lockBastoncino[i].Release();
}
```

La Soluzione 1 è sbagliata!

Può portare allo stallo dei filosofi.

In che modo?

Come evitare lo stallo

Lo stallo può essere evitato se:

a) Usiamo una soluzione asimmetrica distinguendo due classi di filosofi, ad esempio sulla base del loro id (i pari prendono prima la bacchetta destra i dispari quella di sinistra) o ad esempio sull'indice delle bacchette (se la bacchetta di destra ha indice maggiore prendo prima la destra e poi la sinistra altrimenti viceversa). In altre parole si fissa un ordinamento nell'acquisizione delle bacchette per i filosofi.
b) Un filosofo prende le bacchette solo se riesce a prenderle entrambe (le bacchette devono essere acquisite in una sezione critica – «wait *without* holding»)
c) Si usa una soluzione basata sul concetto di «*monitor*» in cui, per l'acquisizione dei bastoncini, si tiene conto dello stato dei filosofi vicini e si agisce di conseguenza

Soluzione che impone un ordinamento nell'acquisizione delle bacchette

Filosofo i

```
while (true) {
      penso();
      if (i % 2) { // filosofo con indice dispari
         // prima acquisisce la bacchetta di sinistra e poi quella di destra
         lockBastoncino[i].Acquire();    lockBastoncino[(i+ 1) mod N].Acquire();
         mangia();  // il filosofo di indice i mangia
         lockBastoncino[(i+ 1) mod N].Release();     lockBastoncino[i].Release();
     } else  { // filosofo con indice pari
         // prima acquisisce la bacchetta di destra e poi quella di sinistra
         lockBastoncino[(i+ 1) mod N].Acquire();     lockBastoncino[i].Acquire();
        mangia();  // il filosofo di indice i mangia
        lockBastoncino[i].Release();    lockBastoncino[(i+ 1) mod N].Release();
    }
}
```

Soluzione che usa tryAcquire
(… ed attesa attiva)

Filosofo i

```
while (true) {
      penso();   // il filosofo pensa
      // il filosofo di indice «i» decide di mangiare
      PrendiBastoncini(&lockBastoncino[i], &lockBastoncino[[(i+ 1) mod N);
      mangia();  // il filosofo di indice «i» mangia
      RilasciaBastoncini(&lockBastoncino[i], &lockBastoncino[[(i+ 1) mod N);
}
```

```
PrendiBasoncini(lock1, lock2) {
  while(true) {
    lock1.Acquire();
    if (lock2.tryAcquire()) return;  // successo
    lock1.Release();       // rilascio il bastoncino e ….
    swap(lock1, lock2); // …. provo nell'ordine contrario
  }
}
```

```
RilasciaBastoncini(lock1, lock2) {
      lock1.Release();
      lock2.Release();  }
```

Soluzione con attesa passiva

Lo stallo può essere evitato se i filosofi adottano una strategia di prevenzione dello stallo, richiedendo con richiesta singola entrambi i bastoncini di cui hanno necessità.

L'assegnazione avverrà se **entrambi** i bastoncini sono disponibili; in caso contrario, il filosofo richiedente si sospenderà senza entrare in possesso di nessun bastoncino, e sarà riattivato da uno dei filosofi che siedono ai suoi lati quando questo rilascerà i propri bastoncini, a condizione che il filosofo sospeso possa ora ottenere entrambi i bastoncini che gli sono necessari.

Il problema

In questa soluzione i filosofi condividono il vettore *stato[i]* (i= 0 .. N-1), dove lo stato del filosofo di indice i può assumere i valori *“HaFame”, “ Mangia”, “Pensa”*, con il seguente significato:

- *stato “HaFame” == > richiede i due bastoncini*
- *stato “Mangia” == > possiede i due bastoncini*
- *transizione “Mangia”→ “Pensa” == > rilascia i due bastoncini*

Si utilizzano inoltre le seguenti variabili:

- Lock mutex: per la mutua esclusione sul vettore *stato;*
- Cond attesaFilosofo[N]: vettore di variabili di condizione utilizzate per la sospensione dei filosofi. La variable attesaFilosofo[i] è usata per l’attesa del filosofo di indice i.

Soluzione che usa un monitor

Filosofo i

```
while (true) {
    penso();   // il filosofo pensa
    // il filosofo di indice i decide di mangiare
    PrendiIBastoncini(i);
    mangia();  // il filosofo di indice i mangia
    RilasciaIBastoncini(i);
}
```

I *monitor* sono meccanismi di sincronizzazione di più alto livello rispetto ai semafori ed alla lock. Possiamo vederli come una collezione di procedure, variabili (di stato e di condizione), strutture dati, confinate all'interno di un modulo. I processi possono chiamare le procedure del monitor (API) senza avere accesso alle sue strutture dati. I monitor sono strutture del linguaggio. Il C non ha i monitor come costrutto nativo ma si possono emulare. Java si: *synchronized*.

PROPRIETA': solamente un processo alla volta è attivo all'interno del monitor.

Soluzione con monitor – filosofo i

Protocollo per mangiare

```
prendiBastoncini(i) {   // metodo del monitor
        // il filosofo di indice i ha deciso di di mangiare
    mutex.Acquire();
    stato[i]= HaFame;
    while  (stato[(i- 1) mod N] == Mangia) || (stato[(i+ 1) mod N] == Mangia )   {
        attesaFilosofo[i].Wait(&mutex);
    }
    stato[i]= Mangia;      // ha ottenuto ambedue i bastoncini
    mutex.Release();
}
```

Soluzione con monitor – filosofo i

Protocollo per pensare

```
rilasciaBastoncini(i) { // metodo del monitor
    mutex.Acquire();
    stato[i]=Pensa;
    if (stato[(i - 1) mod N]== HaFame) && (stato[(i - 2) mod N] != Mangia) {
        // riattiva il filosofo (i-1) mod N se può ottenere entrambi i bastoncini
        stato[(i - 1) mod N = Mangia;
        attesaFilosofo[(i- 1) mod N].Signal(&mutex);
    }
    if (stato[(i + 1) mod N]== HaFame) && (stato[(i + 2) mod N] != Mangia) {
        // riattiva il filosofo (i+1) mod N se può ottenere entrambi i bastoncini
        stato[(i + 1) mod N = Mangia;
        attesaFilosofo[(i + 1) mod N].Signal(&mutex);
    }
    mutex.Release();
}
```

Esercizio

- Proporre una soluzione al problema dei filosofi usando i semafori invece delle variabili di condizione