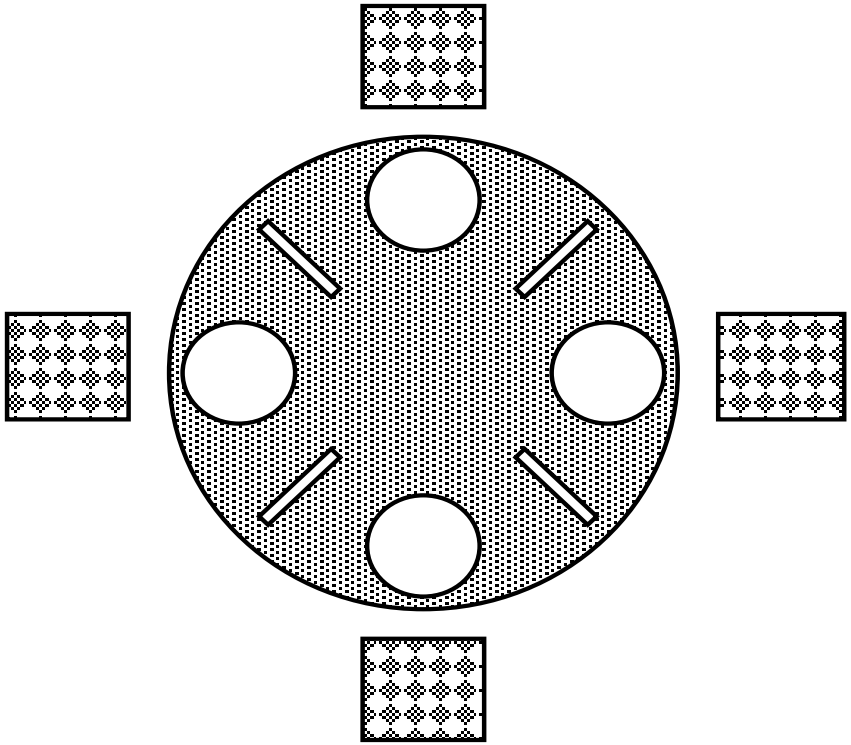
Filosofi a cena

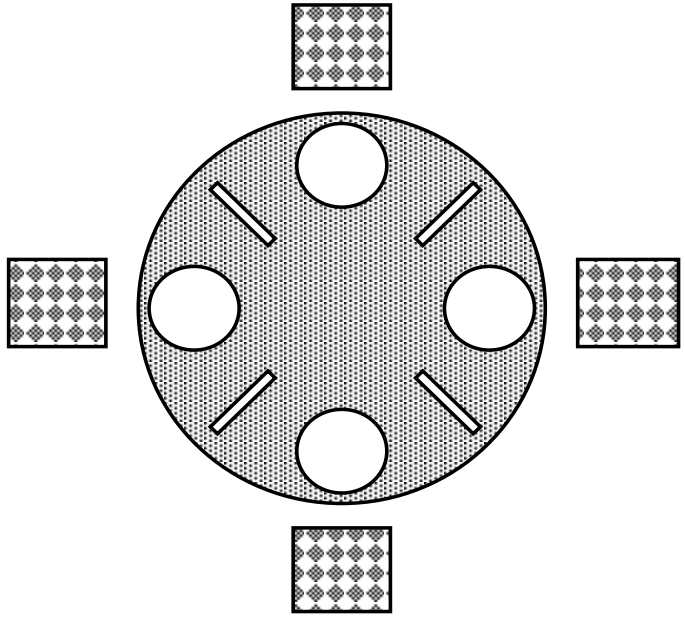
Il problema

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) si sospende 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 si sospende in attesa di ottenerli; dopo aver mangiato torna a pensare rilasciando i due bastoncini.

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

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

Soluzione 1

Filosofo i

```
…
while (true) {
        // il filosofo di indice i decide di mangiare: protocollo per mangiare //
    lockBastoncino[i].Acquire();
        //il filosofo i si sospende se non può ottenere il bastoncino situato alla sua sinistra //
     lockBastoncino[(i+ 1) mod N].Acquire();
        // il filosofo i si sospende se non può ottenere il bastoncino situato alla sua destra //

    < il filosofo di indice i mangia >

        //il filosofo di indice i  ha finito di mangiare: protocollo per pensare //
    lockBastoncino[i].Release();
    lockBastoncino[(i+ 1) mod N].Release();
        // il filosofo rilascia i due bastoncini situati alla sua sinistra e alla sua destra //
}
```

Soluzione 1: sbagliata!

La soluzione 1 può portare allo stallo dei filosofi.

In che modo?

Idee?

Il problema

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 *“ha fame”, mangia”, “pensa”*, con il seguente significato:

- *stato “ha fame” == > 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 – filosofo i

Protocollo per mangiare

```
…
while (true) {
        // il filosofo di indice i decide di mangiare: protocollo per 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();

    < il filosofo di indice i mangia >
        // il filosofo di indice i  ha finito di mangiare: protocollo per pensare //
```

Soluzione – filosofo i

Protocollo per pensare

```
    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();
}
```