Classic synchronization problems

- Bounded buffer problem
- Readers-writer problem
- Dining-philosophers problem
- **The Sleeping Barber problem**

Bounded buffer problem

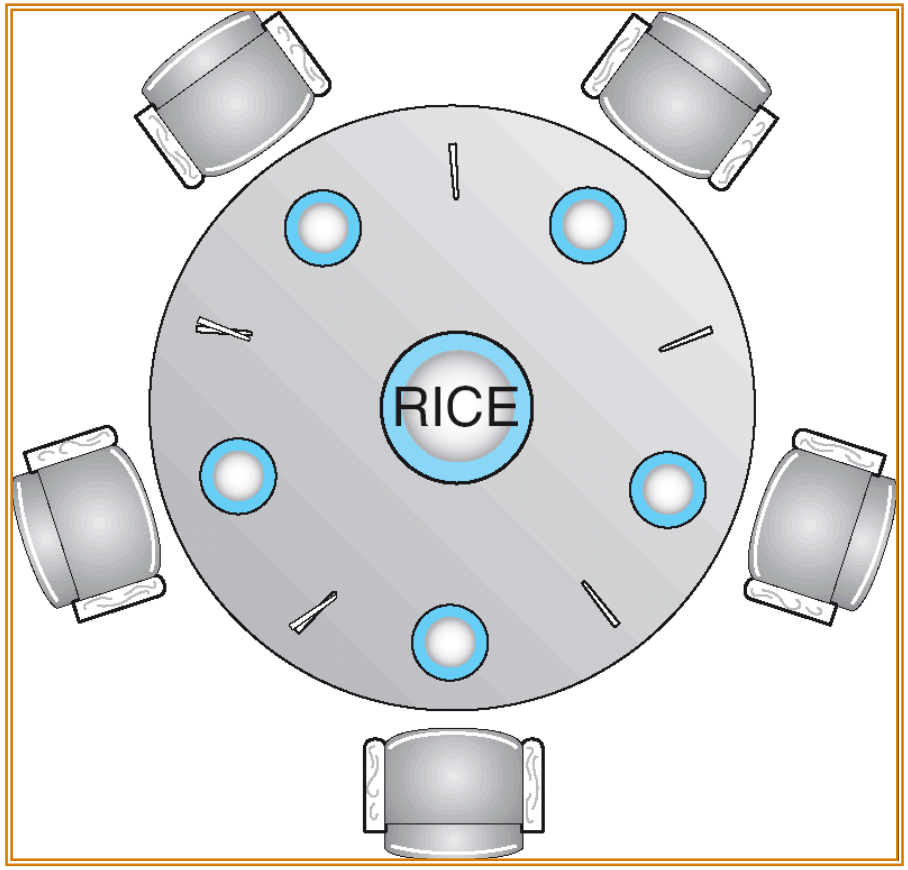
- N element buffer, producer and consumers work with this buffer
- Consumers cannot proceed till producer produced something
- Producer cannot proceed if buffer == N

Reader-writer problem

- Shared database, any number of readers can concurrently read content. Only one writer can write at any one time (with exclusive access)
- Variations:
  - No reader will be kept waiting unless a writer has already received exclusive write permissions
  - Once a writer is ready, it gets exclusive permission as soon as possible. Once a writer is waiting, no further reads are allowed

Dining philosopher's problem

- five philosophers think for some time and then eat
  - Philosophers can only eat if they have both their left and right chopsticks/forks/ at the same time



The Sleeping Barber Problem

- A barbershop consists of a waiting room with `N` chairs, and the barber room containing the barber chair. If there are no customers to be served the barber goes to sleep. If a customer enters the barbershop and all chairs are busy, then the customer leaves the shop. If the barber is busy, then the customer sits in one of the available free chairs. If the barber is asleep, the customer wakes the barber up.

Deadlock and starvation

- Deadlock: processes waiting indefinitely with no chance of making progress
- Starvation: a process waits for a long time to make progress

Semaphore synchronization primitive

- TestAndSet are hard to program for end users
- Introduce a simple function called semaphore:
  - Semaphore is an integer, S
  - Only legal operations on S are:
    - Wait() [atomic] - if S > 0, decrement S else loop
    - Signal() [atomic] - increment S
  - wait (S) {
    while S <= 0
    ; // no-op
    S--;
    }
  - signal (S) {
    S++;
    }
  - Counting (S: is unrestricted), binary (mutex lock) (S: 0, 1)

Semaphore usage example

- Assume synch is initialized to 0
  - P2:

    Wait(synch);

    Statements2;
  - P1:

    Statements1;

    signal(synch);

Monitors

- A high-level abstraction that provides a convenient and effective mechanism for process synchronization
- Only one process may be active within the monitor at a time

```
monitor monitor-name
{
    // shared variable declarations
    procedure P1 (…) { …. }
                …
    procedure Pn (…) {……}
     Initialization code ( ….) { … }
                …
    }
}
```

- In Java, declaring a method *synchronized* to get monitor like behavior
  - What happens to shared variables which are not protected by this monitor?

Condition Variables

- condition x, y;

- Two operations on a condition variable:
  - x.wait () – a process that invokes the operation is suspended.
  - x.signal () – resumes one of processes (if any) that invoked x.wait ()