CS420: Operating Systems Classic Problems of Synchronization

James Moscola
Department of Engineering & Computer Science
York College of Pennsylvania

Classical Problems of Synchronization

- **Classical problems used to test newly-proposed synchronization schemes**
  - Bounded-Buffer Problem
  - Readers-Writers Problem
  - Dining-Philosophers Problem

Bounded-Buffer Problem (Producer/Consumer Problem)

- **In a generalized Bounded Buffer problem, N buffers, each holds one data item**
- **Utilize three semaphores to control buffer access between producer and consumer**
  - Semaphore `mutex` locks access to critical region of code where buffer is modified
    - Initialized to the value 1
  - Semaphore `full` keeps track of how many items are actually in the buffer
    - Initialized to the value 0
  - Semaphore `empty` keeps track of how many available slots there are in the buffer
    - Initialized to the value N

Bounded Buffer Problem (Cont.)

Producer Process

```
do {
    ....
   // produce an item
    ....
    ....
   wait (empty);   // inited to N
   wait (mutex);
     ....
   // add item to the buffer
     ....
   signal (mutex);
   signal (full);
} while (TRUE);
```

Consumer Process

```
do {
    ....
   wait (full);   // full inited to 0
   wait (mutex);
    ....
   // remove item from buffer
    ....
   signal (mutex);
   signal (empty);
    ....
   // consume the item
    ....
 } while (TRUE);
```

Readers-Writers Problems

- **Suitable for testing scenarios where there are multiple readers and multiple writers**
  - e.g. multiple concurrent processes accessing a database, some simply read the database, other write/update the database
- **No problems if multiple readers want to access the data simultaneously**
- **Big problems if any process (reader or writer) attempts to access data while a writer modifies the data**
  - Writers must have exclusive access to the shared data while writing
- **Multiple variations of the Readers-Writers problem exist**
  - Some give preference to readers by providing multiple readers simultaneous access
    - No reader shall be kept waiting if the data is currently available for reading
    - May starve writers
  - Some give preference to writers by putting writers ahead of readers
    - May starve readers

Readers-Writers Problems (First Readers-Writers)

- **First Readers-Writers (preference to readers)**
  - `rw_mutex` protects access to shared data
  - Only first reader needs to lock `rw_mutex`, others bypass if shared data already available for reading
  - `mutex` protects updates to `read_count`
  - If writer is in critical section, only one reader is queued on `rw_mutex`, others are queued on `mutex`

Writer Processes

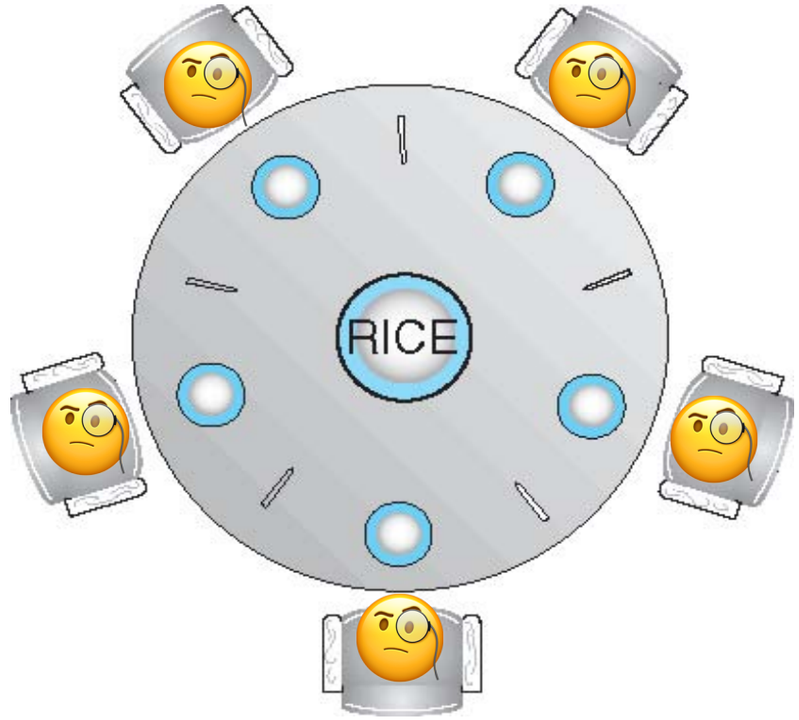
```
while (true) {
  wait(rw_mutex);  // inited to 1
   ....
  // writing is performed
   ....
  signal (rw_mutex);
}
```

Reader Processes

```
while (true) {
  wait(mutex);  // inited to 1
  read_count++;
  if (read_count == 1)
    wait(rw_mutex);
  signal(mutex);
   ....
  // reading is performed (possibly by many)
   ....
  wait(mutex);
  read_count--;
  if (read_count == 0)
    signal(rw_mutex);
  signal(mutex);
}
```

Dining-Philosophers Problem Statement

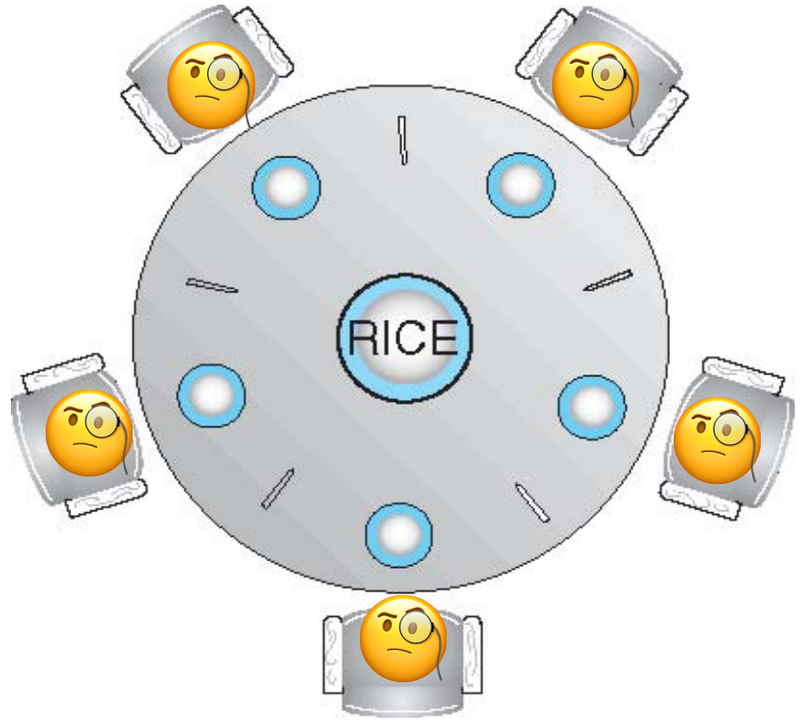
- **Five philosophers spend their lives thinking and eating while sitting at a round table around a bowl of rice**
- **A chopstick is placed between each philosopher**
- **Philosophers cannot interact with their neighbors**
- **Each philosopher will think and occasionally eat**
  - When ready to eat a philosopher will try to pick up 2 chopsticks (one at a time) so he can eat some rice
  - A philosopher needs 2 chopsticks to eat
  - When done eating a philosopher will put down each chopstick, one at a time



- **How can the philosophers sit and eat together without anyone starving?**
  - Think of each chopstick as a semaphore

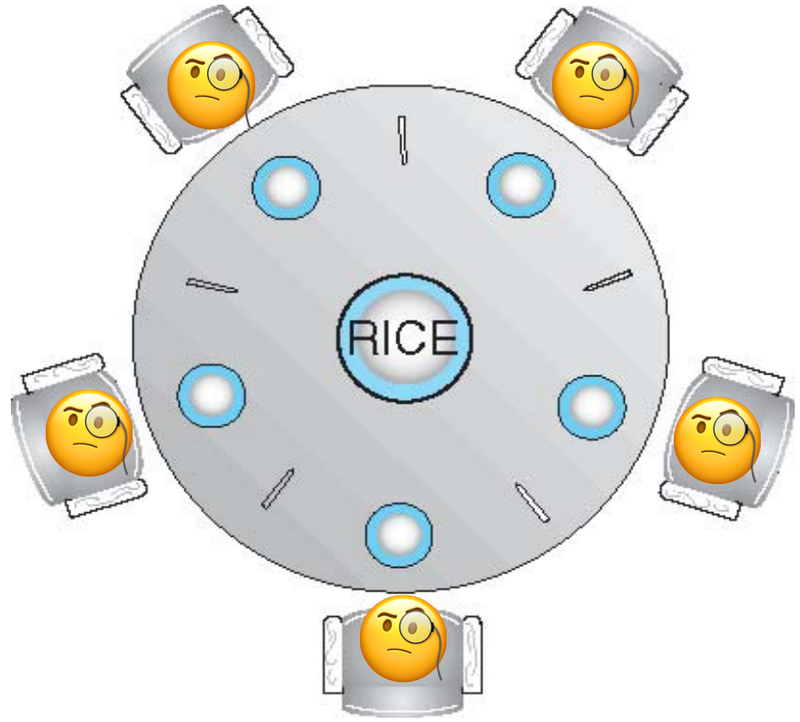
Dining-Philosophers Issues

- **Possible Solution??**
  - Instruct each philosopher to behave as follows:
    - Think until the left chopstick is available; when it is pick it up
    - Think until the right chopstick is available; when it is pick it up
    - Eat some rice
    - Put the left chopstick down
    - Put the right chopstick down
    - Go back to thinking
- **Why might this not work?**



Dining-Philosophers Issues

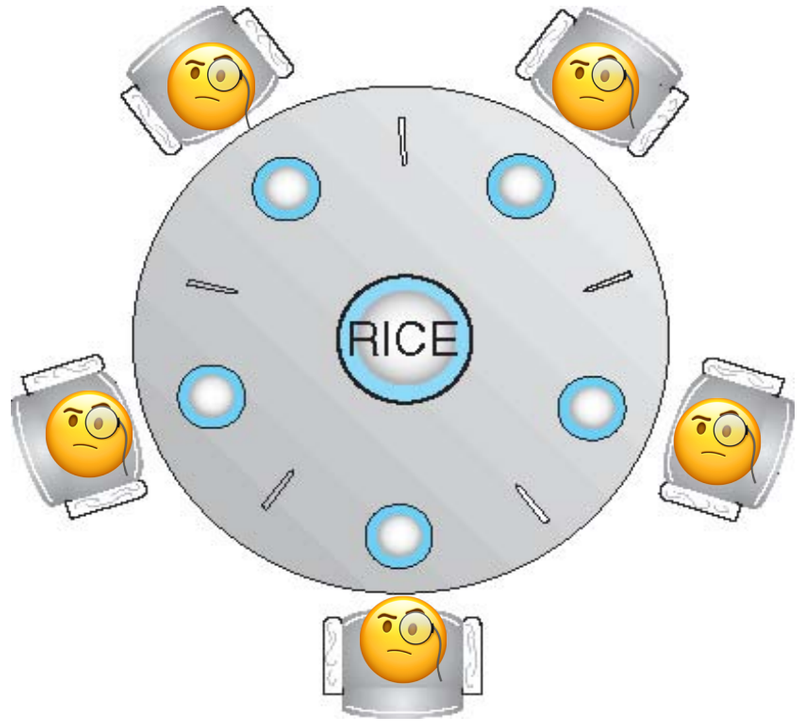
- **Possible Solution??**
  - Instruct each philosopher to behave as follows:
    - Think until the left chopstick is available; when it is pick it up
    - Think until the right chopstick is available; when it is pick it up
    - Eat some rice
    - Put the left chopstick down
    - Put the right chopstick down
    - Go back to thinking



- **Why might this not work?**
  - If each philosopher picks up his left chopstick at the same time, then they all sit waiting for the right chopstick forever (i.e. deadlock ☠)

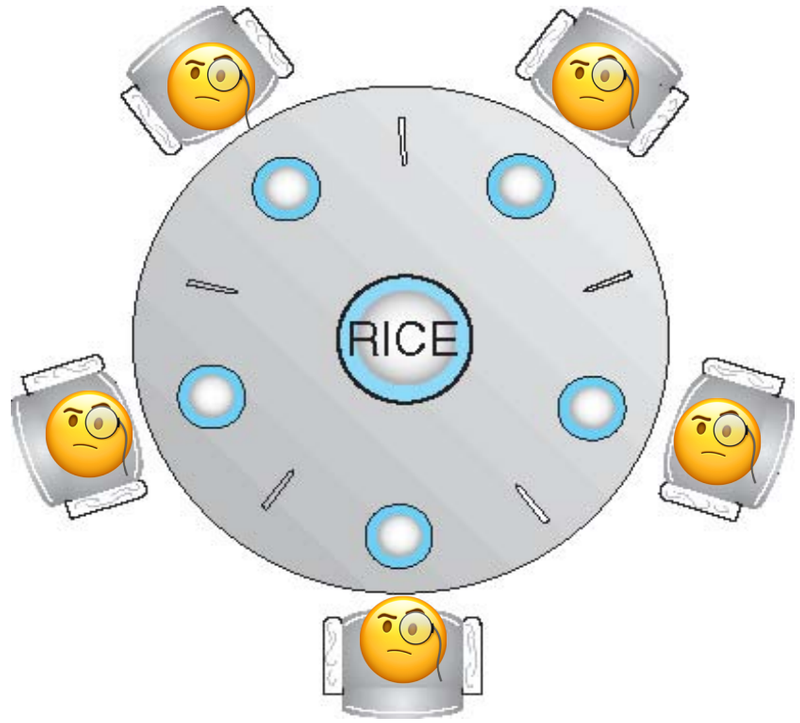
Dining-Philosophers Issues

```
do  {
    wait ( chopstick[i] );
    wait ( chopStick[ (i + 1) % 5] );
       ...
       // eat
       ...
    signal ( chopstick[i] );
    signal ( chopstick[ (i + 1) % 5] );
       ...
       // think
       ...
} while (TRUE);
```



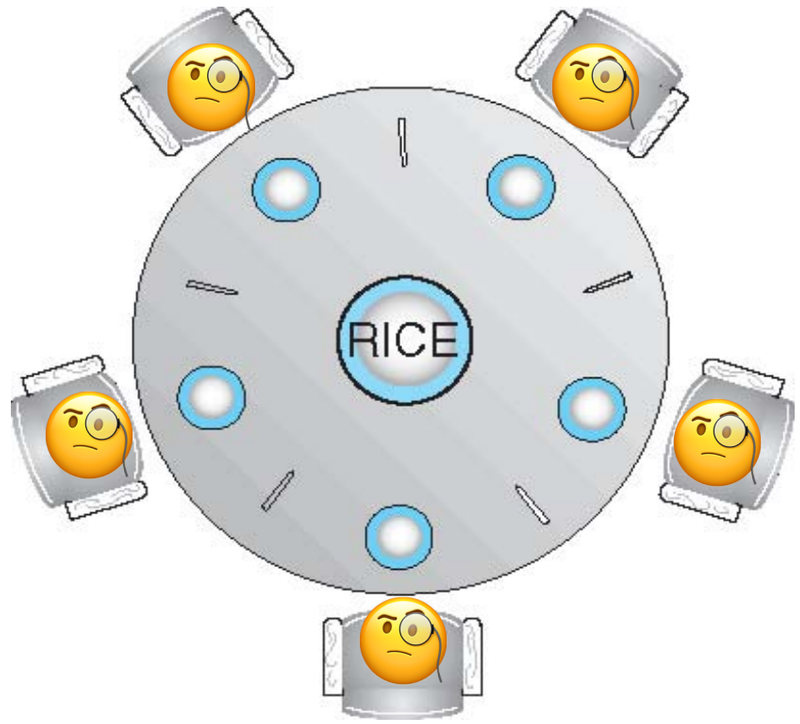
Dining-Philosophers Issues

- **How about telling the philosophers to put down their left chopstick if they've held it for some period of time?**



Dining-Philosophers Issues

- **How about telling the philosophers to put down their left chopstick if they've held it for some period of time?**
  - If the timing is just right, the philosophers may end up in **livelock**
    - The philosophers continue to alternate between thinking and (attempting) to eat, but if the philosophers all try to pick up their chopsticks at the same time, then they will all put them back down at the same time. This cycle will continue until the philosophers starve



Dining-Philosopher Deadlock Remedies

- **Possible remedies to avoid deadlock**
  - Allow at most four philosophers to be sitting at the table simultaneously
  - Allow a philosopher to pick up chopsticks only if both are available (requires an additional mutex to ensure philosopher is not interrupted after picking up the first)
  - Use an asymmetric solution
    - Odd-numbered philosophers pick up left and then right
    - Even-numbered philosophers pick up right and then left