Synchronization III

CS 241

Oct. 16, 2013

Review

- **Mutex**: A simple “lock”
  - pthread_mutex_lock()
  - pthread_mutex_unlock()

- **Conditional Variable**: “monitor” primitive
  - pthread_cond_wait()
  - pthread_cond_signal()
  - pthread_cond_broadcast()

```
void lock() {







}
void unlock() {







}
```

```
void wait() {







}
void post() {







}
```

Semaphore

- A **semaphore** is a “counting” mutex
  - sem_wait()
  - sem_post()

Blocking Bounded Queue (v2)

```
void blocking_queue_push(queue_t *q, void *data) {








   /* queue_push() adds the element to the queue;
      queue_push() is not thread-safe */
   queue_push(q, data);




}
```

Blocking Bounded Queue (v2)

```
void *blocking_queue_pop(queue_t *q) {






   /* queue_pop() pops the top element;
      queue_pop() is not thread-safe */
   void *data = queue_pop(q);






}
```

Deadlock

```
void up() {
  pthread_mutex_lock(&mutex);
  ct++;

}
```

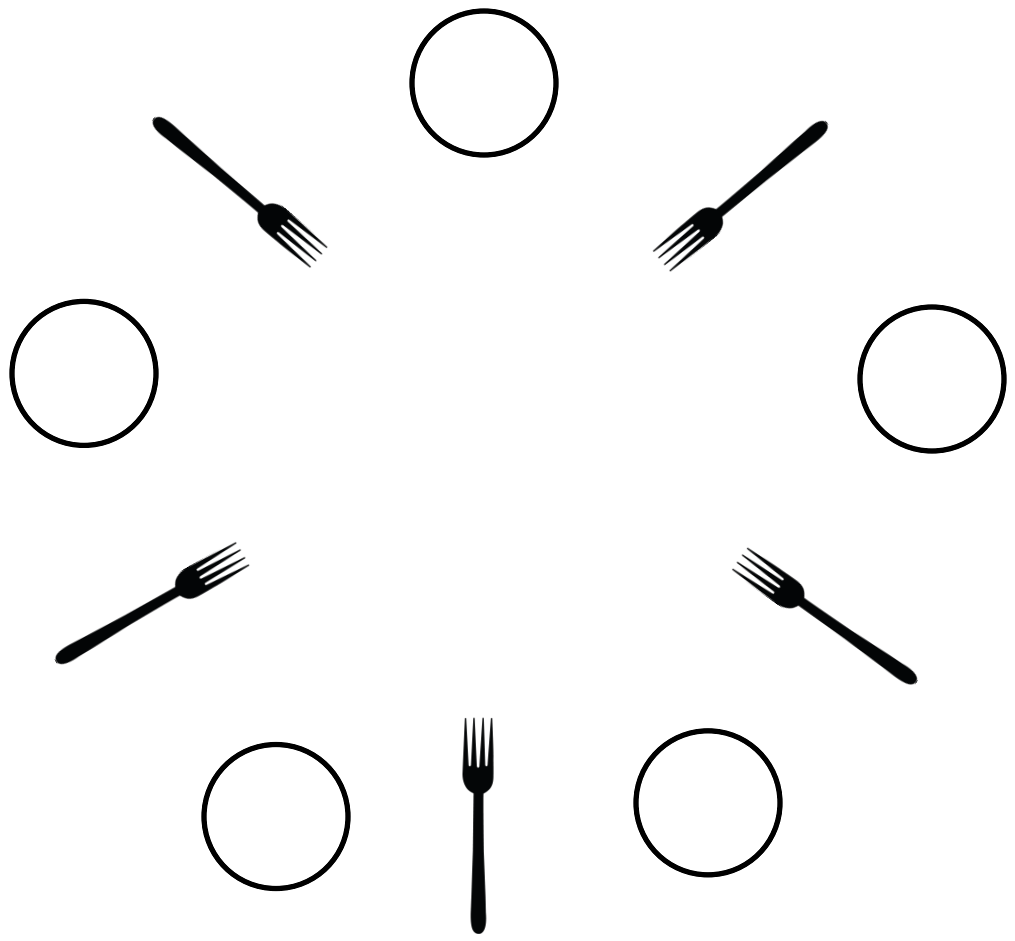
Four Conditions of Deadlock

- In order to guarantee **deadlock**, four conditions <u>**must**</u> be true:
  -
  -
  -
  -

Dinning Philosophers Problem

- **Five philosophers**: Five silent philosophers sit around a table with a bowl of spaghetti.
- **Five forks**: A fork is placed between each pair of adjacent philosophers.
- **Two states**: Philosophers alternate between thinking and eating.
- **Condition**: To eat, a philosopher must have two forks: the fork to his right and the fork to his left.

Deadlock



Mutual Exclusion? ☐

Hold and Wait? ☐

No Preemption? ☐

Circular Wait? ☐