بسم الله الرحمن الرحيم

«سیستم عامل»

١

جلسه ۹: بن‌بست

Resources and deadlocks

- **Processes need access to resources in order to make progress**
- **Examples of computer resources**
  - printers
  - disk drives
  - kernel data structures (scheduling queues …)
  - locks/semaphores to protect critical sections
- **Suppose a process holds resource A and requests resource B**
  - at the same time another process holds B and requests A
  - both are blocked and remain so … this is deadlock

Deadlock modeling: resource usage model

- **Sequence of events required to use a resource**
  - request the resource (like acquiring a mutex lock)
  - use the resource
  - release the resource (like releasing a mutex lock)

- **Must wait if request is denied**
  - block
  - busy wait
  - fail with error code

Preemptable vs nonpreemptable resources

- **Preemptable resources**
  - can be taken away from a process with no ill effects
- **Nonpreemptable resources**
  - will cause the holding process to fail if taken away
  - May corrupt the resource itself
- **Deadlocks occur when processes are granted exclusive access to non-preemptable resources and wait when the resource is not available**

Definition of deadlock

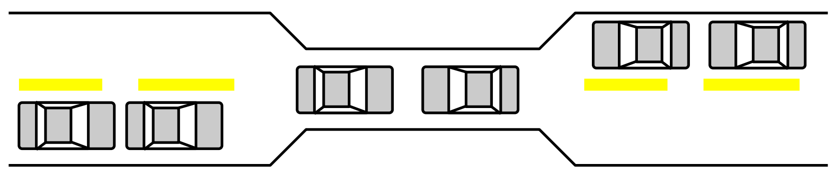
A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause

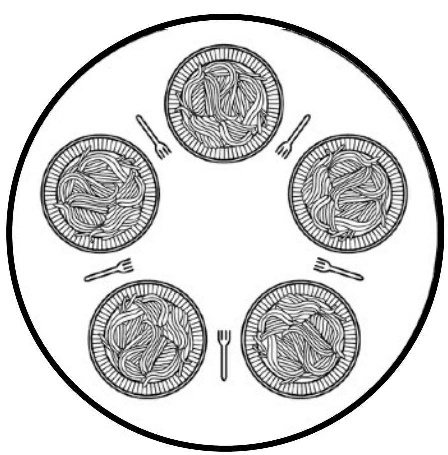
- **The event is the release of a currently held resource**
- **None of the processes can …**
  - be awakened
  - run
  - release resources

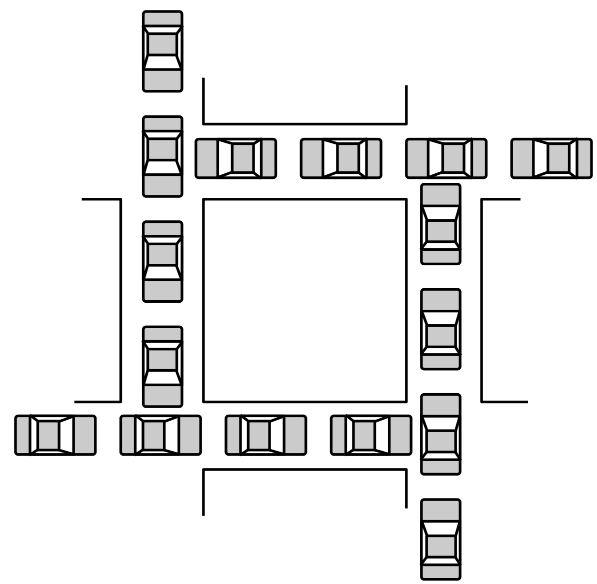
Deadlock conditions

- **A deadlock situation can occur if and only if the following conditions hold simultaneously**
  - Mutual exclusion condition – resource assigned to one process only
  - Hold and wait condition – processes can get more than one resource
  - No preemption condition
  - Circular wait condition – chain of two or more processes (must be waiting for resource from next one in chain)

Examples of deadlock







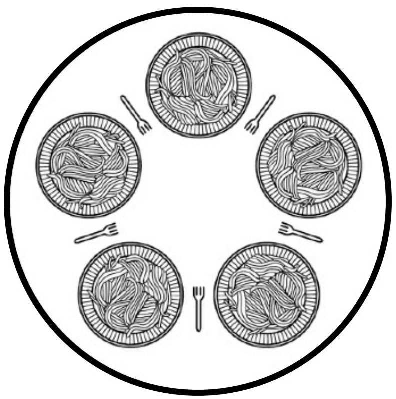
مثالی از کنترل هم‌روندی:

غذای فیلسوفان

Dining philosophers problem

- Five philosophers sit at a table
- One chopstick between each philosopher (need two to eat)

Each philosopher is modeled with a thread



```
while(TRUE) {
  Think();
  Grab first chopstick;
  Grab second chopstick;
  Eat();
  Put down first chopstick;
  Put down second chopstick;
}
```

- Why do they need to synchronize?
- How should they do it?

Is this a valid solution?

```
#define N 5

Philosopher(i) {
  while(TRUE) {
    Think();
    take_chopstick(i);
    take_chopstick((i+1)% N);
    Eat();
    put_chopstick(i);
    put_chopstick((i+1)% N);
  }
}
```

Problems

- Potential for deadlock !

Working towards a solution …

```
#define N 5

Philosopher() {
   while(TRUE) {
      Think();
      take_chopstick(i);
      take_chopstick((i+1)% N);
      Eat();
      put_chopstick(i);
      put_chopstick((i+1)% N);
   }
}
```

take_chopsticks(i)

put_chopsticks(i)

Working towards a solution …

```
#define N 5

Philosopher() {
  while(TRUE) {
    Think();
    take_chopsticks(i);
    Eat();
    put_chopsticks(i);
  }
}
```

Taking chopsticks

```
int state[N]
semaphore mutex = 1
semaphore sem[i]
```

```
take_chopsticks(int i) {
  wait(mutex);
  state [i] = HUNGRY;
  test(i);
  signal(mutex);
  wait(sem[i]);
}
```

```
// only called with mutex set!

test(int i) {
 if (state[i] == HUNGRY &&
      state[LEFT] != EATING &&
      state[RIGHT] != EATING){
   state[i] = EATING;
   signal(sem[i]);
 }
}
```

Putting down chopsticks

```
int state[N]
semaphore mutex = 1
semaphore sem[i]
```

```
put_chopsticks(int i) {
  wait(mutex);
  state [i] = THINKING;
  test(LEFT);
  test(RIGHT);
  signal(mutex);
}
```

```
// only called with mutex set!

test(int i) {
 if (state[i] == HUNGRY &&
      state[LEFT] != EATING &&
      state[RIGHT] != EATING){
    state[i] = EATING;
    signal(sem[i]);
 }
}
```

Dining philosophers

- Is the previous solution correct?
- What does it mean for it to be correct?
- Is there an easier way?

Resource acquisition scenarios

Thread A:

```
acquire (resource_1)
use resource_1
release (resource_1)
```

Example:

```
var r1_mutex: Mutex
...
r1_mutex.Lock()
```
Use resource_1
```
r1_mutex.Unlock()
```

Resource acquisition scenarios

Thread A:

```
acquire (resource_1)
use resource_1
release (resource_1)
```

Another Example:

```
var r1_sem: Semaphore
r1_sem.Signal()
...
r1_sem.Wait()
```
Use resource_1
```
r1_sem.Signal()
```

Resource acquisition scenarios

Thread A:

```
acquire (resource_1)
use resource_1
release (resource_1)
```

Thread B:

```
acquire (resource_2)
use resource_2
release (resource_2)
```

Resource acquisition scenarios

Thread A:

```
acquire (resource_1)
use resource_1
release (resource_1)
```

Thread B:

```
acquire (resource_2)
use resource_2
release (resource_2)
```

No deadlock can occur here!

Resource acquisition scenarios: 2 resources

Thread A:

```
acquire (resource_1)
acquire (resource_2)
use resources 1 & 2
release (resource_2)
release (resource_1)
```

Thread B:

```
acquire (resource_1)
acquire (resource_2)
use resources 1 & 2
release (resource_2)
release (resource_1)
```

Resource acquisition scenarios: 2 resources

Thread A:

```
acquire (resource_1)
acquire (resource_2)
use resources 1 & 2
release (resource_2)
release (resource_1)
```

Thread B:

```
acquire (resource_1)
acquire (resource_2)
use resources 1 & 2
release (resource_2)
release (resource_1)
```

No deadlock can occur here!

Resource acquisition scenarios: 2 resources

Thread A:

```
acquire (resource_1)
use resources 1
release (resource_1)
acquire (resource_2)
use resource 2
release (resource_2)
```

Thread B:

```
acquire (resource_2)
use resources 2
release (resource_2)
acquire (resource_1)
use resource 1
release (resource_1)
```

Resource acquisition scenarios: 2 resources

Thread A:

```
acquire (resource_1)
use resources 1
release (resource_1)
acquire (resource_2)
use resource 2
release (resource_2)
```

Thread B:

```
acquire (resource_2)
use resources 2
release (resource_2)
acquire (resource_1)
use resource 1
release (resource_1)
```

No deadlock can occur here!

Resource acquisition scenarios: 2 resources

Thread A:

```
acquire (resource_1)
acquire (resource_2)
use resources 1 & 2
release (resource_2)
release (resource_1)
```

Thread B:

```
acquire (resource_2)
acquire (resource_1)
use resources 1 & 2
release (resource_1)
release (resource_2)
```

Resource acquisition scenarios: 2 resources

Thread A:

```
acquire (resource_1)
acquire (resource_2)
use resources 1 & 2
release (resource_2)
release (resource_1)
```

Thread B:

```
acquire (resource_2)
acquire (resource_1)
use resources 1 & 2
release (resource_1)
release (resource_2)
```

Deadlock is possible!