Operating Systems

0107451

Chapter 2 Processes and Threads

Dr. Naeem Odat

Tafial Technical University
Department of Computer and Communications Engineering

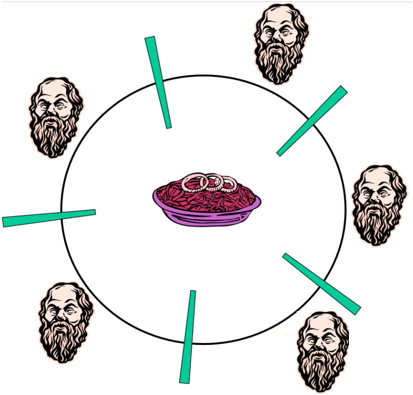
2.3 Dining Philosophers Problem

Dining Philosophers

- Classic Synchronization Problem
- Philosopher
    - eat, think, sleep
    - eat, think, sleep
    - . . . . . .
- Philosopher = Process
- Eating needs two resources (chopsticks)

2.3 Dining Philosophers Problem

Dining Philosophers



Problem

Each philosopher needs two chopsticks to eat.

2.3 Dining Philosophers Problem

First pass at a solution

One Mutex for each chopstick

```
Philosopher i:
    while (1):
        Think();
        lock(Left_Chopstick);
        lock(Right_Chopstick);
        Eat();
        unlock(Left_Chopstick);
        unlock(Right_Chopstick);
```

2.3 Dining Philosophers Problem

2.3 Dining Philosophers Problem

2.3 Dining Philosophers Problem

2.3 Dining Philosophers Problem

One possible solution

Use a mutex for the whole dinner-table

```
Philosopher i:
    lock(table);
    Eat();
    Unlock(table);
```

Problem?

Performance problem

2.3 Dining Philosophers Problem

Another solution

```
Philosopher i:
    Think();
    unsuccessful = 1;
    while (unsuccessful):
        lock(left_chopstick);
        if(try_lock(right_chopstick)) /*returns immediately if
                                         unable to grab the lock */
            unsuccessful = 0;
        else
            unlock(left_chopstick);

    Eat();

    unlock(left_chopstick);
    unlock(right_chopstick);
```

Problems?

Starvation if unfavorable scheduling!

2.3 Dining Philosophers Problem

In practice

- Starvation will probably not occur
- We can ensure this by adding randomization to the system:
  - Add a random delay before retrying.
  - Unlikely that our random delays will be in sync too many times.

2.3 Dining Philosophers Problem

Solution with random delays

```
Philosopher i:
    Think();
    unsuccessful = 1;
    while (unsuccessful):
        wait(random());
        lock(left_chopstick);
        if(try_lock(right_chopstick))
            unsuccessful = 0;
        else
            unlock(left_chopstick);

    Eat();

    unlock(left_chopstick);
    unlock(right_chopstick);
```

2.3 Dining Philosophers Problem

Another solution

Suppose we have two philosophers

```
Philosopher 1:
    lock(left_chopstick);
    lock(right_chopstick);

Philosopher 2:
    lock(right_chopstick);
    lock(left_chopstick);
```

Does this work?

Does it work for 3 philosophers? 4? 5? ...

2.3 Dining Philosophers Problem

Yet another solution

- Do not try to take forks one after another
- Don't have each fork protected by a different mutex
- Try to grab both forks at the same time

Text has details