Concurrency Bugs CS 411

Atomicity Violation

Use locks around critical sections!

Order Violation

- Ensure the shared object exist/is initialized before use.

One way to accomplish this is with a barrier implemented with a mutex and condition variable

Pseudo-Code

```
mutex m;
cond_var c;
int go_ahead = 0;
```

Main thread:

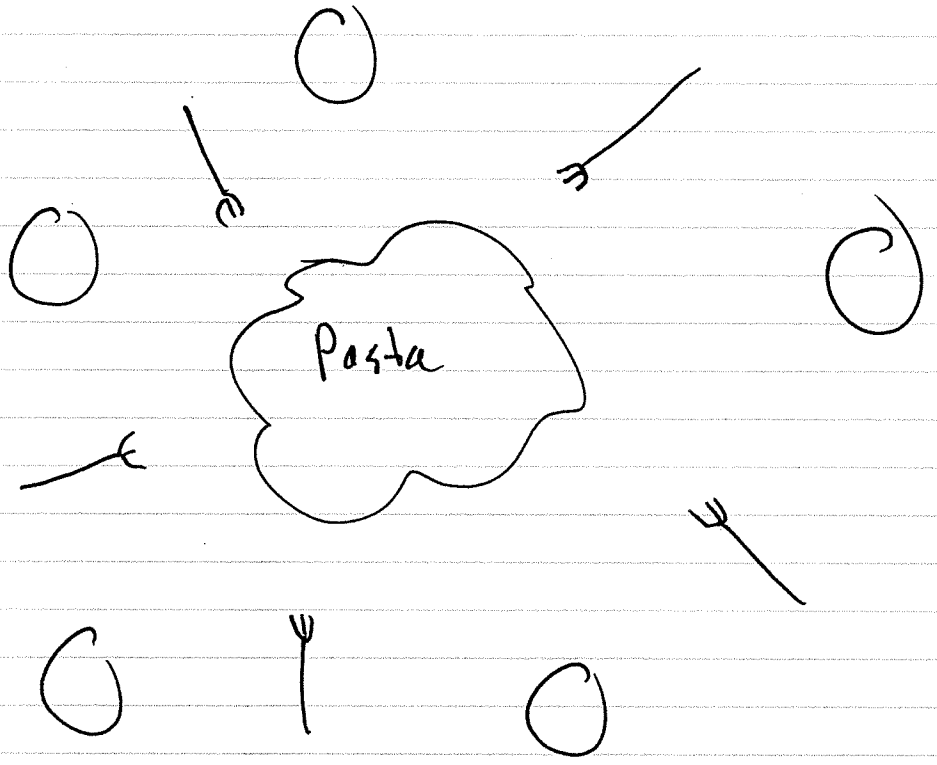
```
...
lock(m);
go_ahead = 1;
cond_broadcast(c);
unlock(m);
```

Other threads:

```
lock(m);
while(!go_ahead)
    cond_wait(c);
unlock(m);
```

Deadlock

Dining Philosophers



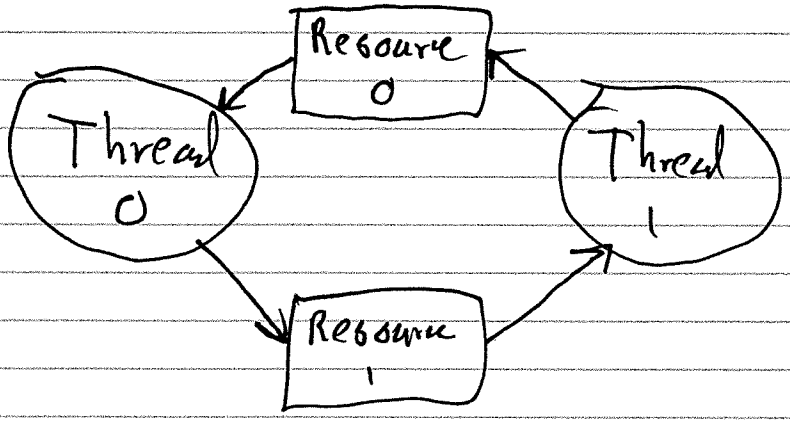
Philosophers alternate between thinking + eating
(non-cs) (cs)

Must acquire 2 adjacent forks to eat.

What can go Wrong?

Necessary Conditions for Deadlock:

- Mutual Exclusion
  Resource can only be used by 1 thread
- Hold-and-Wait
  thread holds resources while trying to acquire others
- No preemption
  - Resources can't be taken away from a thread
- Circular Wait



Deadlocks can be <u>prevented</u> by ensuring that 1 of the necessary conditions don't hold

Dining Philosophers

Deadlock Avoidance

Banker's Algorithm
- can be hard to apply in practice

Detect and Recover

Example: Database Transactions

Exercise

Using the semaphore-based solution to producer-consumer:

1) How are atomicity and order violations avoided

2) We saw deadlock occur when we changed the order of the semaphore waits. With the correct ordering, which of the 4 deadlock necessary conditions is prevented from occurring?