COMP31212: Concurrency
Topics 5.1: Deadlock
Outline

**Topic 5.1: Deadlock**
- What is Deadlock?
- Examples
- Dining Philosophers
- Livellock
Outline

Topic 5.1: Deadlock

What is Deadlock?
Examples
Dining Philosophers
Livelock
What is Deadlock?

What is it?

In a concurrent system:

- no (FSP) process has a following action
- there is a (LTS) state from which no action is possible
- all (Java) threads blocked/waiting
Four necessary and sufficient conditions:

1. Serially reusable resources: the processes share resources which they use under mutual exclusion.

Four necessary and sufficient conditions:

1. Serially reusable resources: the processes share resources which they use under mutual exclusion.

2. Incremental acquisition: processes hold on to resources already allocated to them while waiting to acquire additional resources.

Four **necessary and sufficient conditions:**

1. **Serially reusable resources:** the processes share resources which they use under mutual exclusion.
2. **Incremental acquisition:** processes hold on to resources already allocated to them while waiting to acquire additional resources.
3. **No preemption:** once a resource is acquired, other processes cannot gain access until it is released.

Four **necessary and sufficient conditions:**

1. **Serially reusable resources:** the processes share resources which they use under mutual exclusion.

2. **Incremental acquisition:** processes hold on to resources already allocated to them while waiting to acquire additional resources.

3. **No preemption:** once a resource is acquired, other processes cannot gain access until it is released.

4. **Wait-for cycle:** a cycle of processes exists such that each process can hold a resource which its successor in the cycle is waiting for.

Outline

Topic 5.1: Deadlock

What is Deadlock?
Examples
Dining Philosophers
Livelock
FSP Example

A simple example of printer and scanner use.

RESOURCE = (get -> put -> RESOURCE).

P = (printer.get -> scanner.get -> copy ->
   printer.put -> scanner.put -> P).
Q = (scanner.get -> printer.get -> copy ->
   scanner.put -> printer.put -> Q).

||SYSTEM = (p:P || q:Q ||
   {p,q}::printer:RESOURCE ||
   {p,q}::scanner:RESOURCE).

This can deadlock. How?
Topic 5.1: Deadlock

Outline

Topic 5.1: Deadlock
What is Deadlock?
Examples
Dining Philosophers
Livelock
The “Dining Philosophers” Example

Classic example due to Dijkstra.

$N$ philosophers sit around a circular table
The “Dining Philosophers” Example

Classic example due to Dijkstra.

$N$ philosophers sit around a circular table
Each philosopher alternately “thinks” then “eats”
The “Dining Philosophers” Example

Classic example due to Dijkstra.

$N$ philosophers sit around a circular table
Each philosopher alternately “thinks” then “eats”
Unending supply of tangled spaghetti at table centre
The “Dining Philosophers” Example

Classic example due to Dijkstra.

\( N \) philosophers sit around a circular table
Each philosopher alternately “thinks” then “eats”
Unending supply of tangled spaghetti at table centre
Philosophers are poor, only \( N \) forks available, each one placed between two philosophers
The “Dining Philosophers” Example

Classic example due to Dijkstra.

$N$ philosophers sit around a circular table
Each philosopher alternately “thinks” then “eats”
Unending supply of tangled spaghetti at table centre
Philosophers are poor, only $N$ forks available, each one placed between two philosophers
Each philosopher needs two forks to eat, those to the immediate left and right
The “Dining Philosophers” Example

Classic example due to Dijkstra.

$N$ philosophers sit around a circular table
Each philosopher alternately “thinks” then “eats”
Unending supply of tangled spaghetti at table centre
Philosophers are poor, only $N$ forks available, each one placed between two philosophers
Each philosopher needs two forks to eat, those to the immediate left and right
No philosopher must go hungry
Some Dining Philosophers
Some Dining Philosophers
Dining Philosophers - aspects

- concurrently executing processes
Dining Philosophers - aspects

- concurrently executing processes
- non-critical independent execution
Dining Philosophers - aspects

- concurrently executing processes
- non-critical independent execution
- simultaneous access to shared, serially reusable, resources
Dining Philosophers - aspects

- concurrently executing processes
- non-critical independent execution
- simultaneous access to shared, serially reusable, resources
- deadlock: incremental acquisition, no preemption, wait-for cycle
Dining Philosophers - aspects

- concurrently executing processes
- non-critical independent execution
- simultaneous access to shared, serially reusable, resources
- deadlock: incremental acquisition, no preemption, wait-for cycle

- several solutions: any solution must deny at least one of the 4 conditions for deadlock. Example: introduce a ‘butler’ to schedule the philosophers, so that no cycle can occur.
Dining Philosophers - aspects

- concurrently executing processes
- non-critical independent execution
- simultaneous access to shared, serially reusable, resources
- deadlock: incremental acquisition, no preemption, wait-for cycle

- several solutions: any solution must deny at least one of the 4 conditions for deadlock. Example: introduce a ‘butler’ to schedule the philosophers, so that no cycle can occur.
- fairness issues
Dining Philosophers in FSP

FORK = (get -> put -> FORK).

PHIL = (sitdown->right.get->left.get
       ->eat->left.put->right.put
       ->getup->PHIL).

||DINERS(N=5)=
   forall [i:0..N-1]
       (phil[i]:PHIL
        ||{phil[i].left,phil[((i-1)+N) mod N].right}::FORK).

where E mod N is the remainder after E is divided by N (this makes the sharing cyclic around the table).
Dining Philosophers in Java - Fork

class Fork {

    private boolean taken=false;

    synchronized void put() {
        taken=false;
        notify();
    }

    synchronized void get() throws InterruptedException {
        while (taken) wait();
        taken=true;
    }

}
Dining Philosophers in Java - Philosopher

class Philosopher extends Thread {
    private Fork left; private Fork right;

    Philosopher(... Fork l, Fork r) {
        ... left = l; right = r; ...
    }

    public void run() {
        try {
            while (true) {
                // thinking...
                // hungry:
                right.get(); left.get();
                // eating...
                // finished eating:
                right.put(); left.put();
            }
        } catch (InterruptedException e) {}}
}
Outline

Topic 5.1: Deadlock
What is Deadlock?
Examples
Dining Philosophers
Livelock
Livelock

A concurrent system may be able to perform actions, but nevertheless may not be able to make progress towards a goal. Such a situation is called **livelock**.

**Example:** two polite people trying to go through an open door in the opposite directions. Each makes a step towards the door, but retracts on seeing the other trying to enter.

To analyse livelock, we need a notion of *progress*. We look at this later.