Mutex Lock Problems

- Deadlocks
- Starvation
- Livelock
A set of threads is **deadlocked** if each thread in the set holds a lock and is waiting for a lock that some other thread in the set holds.

- So no thread can run!

Deadlocks are forever!

- Breaking them generally requires killing threads.
A deadlock can only occur under these conditions:

- **Mutual exclusion** – resources cannot be shared by threads
- **Hold and wait** – threads hold a resource and then wait to acquire other resources
- **No preemption** – acquired resources cannot be preempted
- **Circular wait** – threads form a circular chain, each waiting for a resource from the next thread in the chain
Examples of Deadlock

Mahjong
Gridlock
Detecting Deadlocks

- Deadlocks last forever, need to detect them
- Use circular wait condition
- Represent chain of threads using **wait-for** graph

```plaintext
Thread_A() {
    lock(R1);
    lock(R2);
    use resource 1 and 2;
    unlock(R1);
    unlock(R2);
}

Thread_B() {
    lock(R2);
    lock(R1);
    use resource 1 and 2;
    unlock(R2);
    unlock(R1);
}
```

- Deadlock ⇔ Cycle in the wait-for graph
Preventing Deadlocks

- **Avoid hold and wait**
  - If a lock is unavailable, release previously acquired locks, and try to reacquire all locks again
  - Problems with this approach?
    - Threads may not make progress
    - Need to undo changes before releasing locks

- **Prevent circular wait**
  - Number each of the resources
  - Require each thread to acquire lower numbered resources before higher numbered resources
  - Problems with this approach?
    - Hard to number all resources
    - Hard to ensure that resources are acquired in correct order
Deadlock, Starvation, Livelock

- **Deadlock**
  - A set of threads perform no work due to circular wait
  - Once a deadlock occurs, it does not go away

- **Starvation**
  - A thread performs no work because the resources it needs are being used by others constantly
    - E.g., newly arriving high-priority thread acquires lock before waiting low-priority thread can acquire lock
  - Starvation can be a temporary condition

- **Livelock**
  - A thread continues to run but makes no progress!
    - E.g., deadlock prevention by avoiding hold and wait
  - Livelock can be a temporary condition
Summary

- Mutual exclusion is implemented using locks
- Finer-grained locks provide more concurrency
- However, fine-grained locks can lead to several locking problems
  - Deadlock: no progress
  - Starvation: severe performance degradation
  - Livelock: severe performance degradation

- Next lecture discusses synchronization
Think Time

- Are there systems that can handle deadlocks automatically?
- How can one avoid starvation?
- How can one avoid livelock?
- Look up interrupt storm on wikipedia. Does this event cause a deadlock, starvation or livelock? How can be solved?
Are there systems that can handle deadlocks automatically?

- Yes, databases often handle deadlocks automatically. Databases acquire locks for applications automatically and the locking order may cause cycles, which lead to deadlocks. Databases use a heavy-weight machinery called transactions to abort and rollback transactions when a deadlock is detected. This abort and rollback prevents deadlocks by avoiding the no-preemption condition required for deadlocks.
Think Time Answers

- How can one avoid starvation?
  - High priority threads should not be allowed to run for long periods because they can starve low priority threads. Within the same priority threads, we can use fifo (first-come, first-served) queuing whenever a thread needs to wait on a condition (e.g., while trying to acquire a lock), so the first thread to wait on the condition gets served first (e.g., the lock implementation can use a wait queue from which threads acquire locks in fifo order).

- How can one avoid livelock?
  - One method is to ensure that threads run for a while before switching to running some other thread. This way, the thread gets to make some progress, before other threads run.
Look up interrupt storm on wikipedia. Does this event cause a deadlock, starvation or livelock? How can be solved?

- Interrupt storm can cause a livelock. The solution to an interrupt storm is to disable interrupts and switch to polling when interrupts are arriving too often. This is similar to receiving too many popups (interrupts) for email messages, which leads to no work being done (livelock). At some point, the user may disable the popups, and then periodically check their mail inbox (polling).