Deadlock

COMP755 Advanced Operating Systems

Deadlock Definitions

- A deadlock occurs when two or more tasks are waiting for each other and they cannot proceed.
- Deadlock is a situation where two or more processes are each waiting for a resource that another process in the group holds.
- Deadlock is the permanent blocking of a set of processes.
- Deadlock occurs when waiting processes cannot be removed from the wait state.

Hung Systems

- Deadlock typically results in “hung” systems. The program is running, but nothing is happening.
- If there is a possible order of execution that will allow all tasks to complete, then the system is not deadlocked.

Deadlock Example

- Consider a program with two threads. Each thread requires exclusive access to the resources A and B.

Thread 1	Thread 2
Acquire resource A	Acquire resource B
Acquire resource B	Acquire resource A

Both threads are deadlocked.

Necessary Conditions for Deadlock

1. Mutual Exclusion
2. No Preemption
3. Hold and Wait
4. Circular wait

Mutual Exclusion

- Deadlock will only occur if a resource cannot be simultaneously used by more than one task.
- Some resources can easily be shared
  - Read only files
  - Code with no shared data

No Preemption

- A higher priority thread cannot “steal” a resource from a lower priority thread.
- Some resources, such as the CPU, can be easily preempted and given to another thread. The dispatcher frequently suspends a thread and runs another.
- If one thread can take resources from another, then it can grab all it needs and run to completion.

Hold and Wait

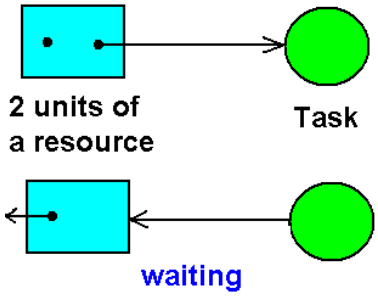
- A thread has to have a shared resource allocated to it and be waiting for another resource.
- If a thread gets all of its necessary resources at once, then it will not deadlock.
- A thread has to have one resource and be waiting for more to deadlock.

Circular Wait

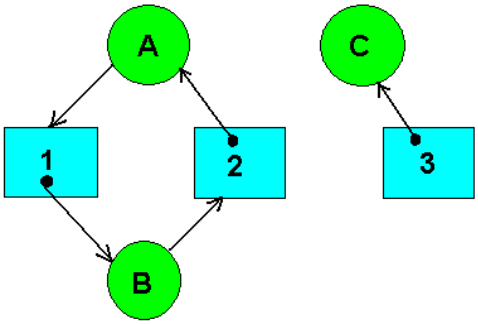
- The resource diagram has to have a cycle, a path following the directed edges that starts at one point and returns to the same point.
- A deadlock can not occur unless the graph of tasks and resources contains a cycle.

Circular Wait Graph

- Tasks are circles and resources are boxes with one dot for each unit of resource.



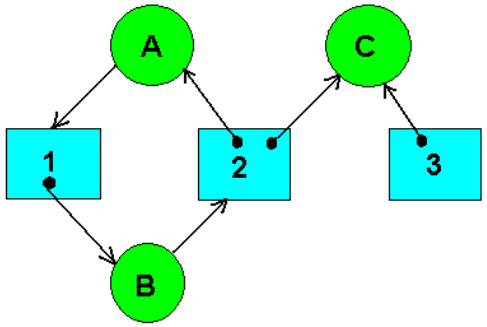
Deadlock Cycle



Necessary but Not Sufficient

- The four requirements for deadlock:
  1. Mutual Exclusion
  2. No Preemption
  3. Hold and Wait
  4. Circular wait
- These must be present for a deadlock to occur, but their presence does not necessarily mean there must be a deadlock.

Non-Deadlock Cycle



Necessary and Sufficient Conditions for Deadlock

- 1 - 3 as before
- A **Knot** in the resource graph
- A Knot exists when the reachable set of nodes is equal to exactly that set of nodes.

Starvation

- Starvation occurs when a running task is indefinitely prevented from accessing a resource.
- It differs from deadlock in that involved tasks are running.

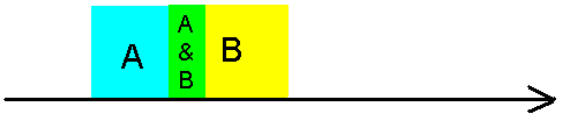
Handling Deadlock

1. Ignore deadlocks until they occur, then terminate tasks until the problem goes away.
2. Avoid entering into a state that might deadlock. Banker's algorithm does this.
3. Design the system so that deadlock cannot occur.

Execution Timeline

The program acquires resource A, releases it and then acquires resource B.

Execution Timeline

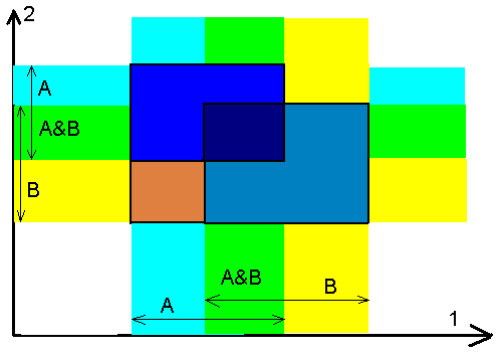


The program acquires resource A then resource B before releasing A.

Multiple Thread Timelines

- Consider the execution timelines of two threads drawn with the flow of thread 1 perpendicular to the flow of thread 2.
- In the following example, thread 1 acquires resource A then resource B while thread 2 wants resource B then resource A.

Unsafe Zone



Deadlock Proofing

- Design the system so that at least one of the necessary condition cannot hold.
- When proving a system cannot deadlock, show that one of the conditions cannot hold.