Operating Systems:

Chap 6:

DeadLocks: Conditions and Handling

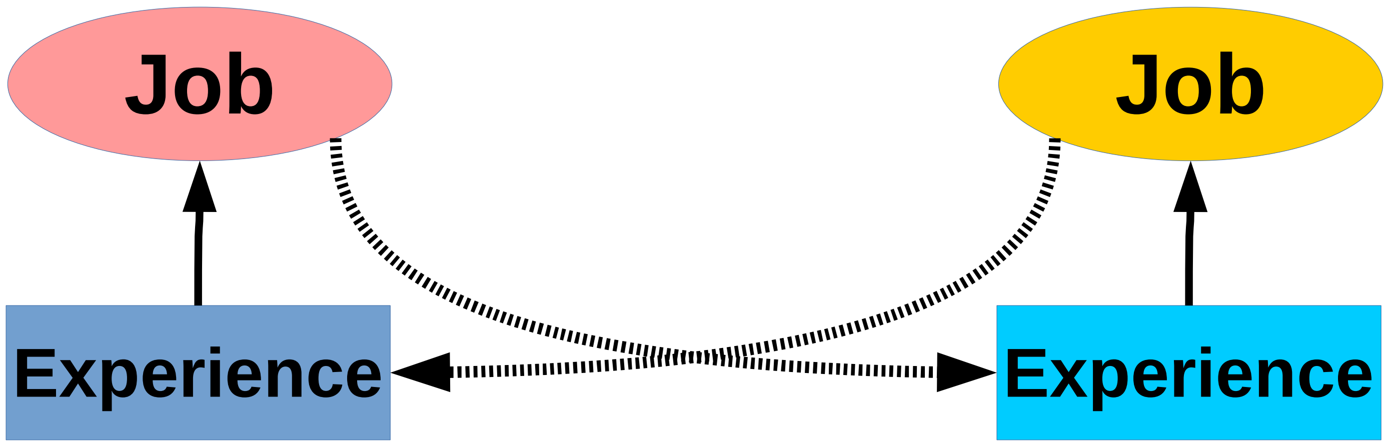
CS400

Week 13: 8th April

Spring 2020

Oliver BONHAM-CARTER

Deadlocks in the Human World



- To make money, you have to spend money
- To get a job, you must have experience, to get experience, you must have a job.
- The chicken must have come before the egg, yet the egg must have come before the chicken.

Handling Deadlocks

- Deadlock Prevention and avoidance: Make sure that deadlock can never happen
- Prevention: Ensure that each of the four *deadlock conditions* (discussed later on) is understood and does not cause system failure.
- Avoidance: The OS needs more information so that is can determine if the current request can be satisfied or delayed.

Handling Deadlocks

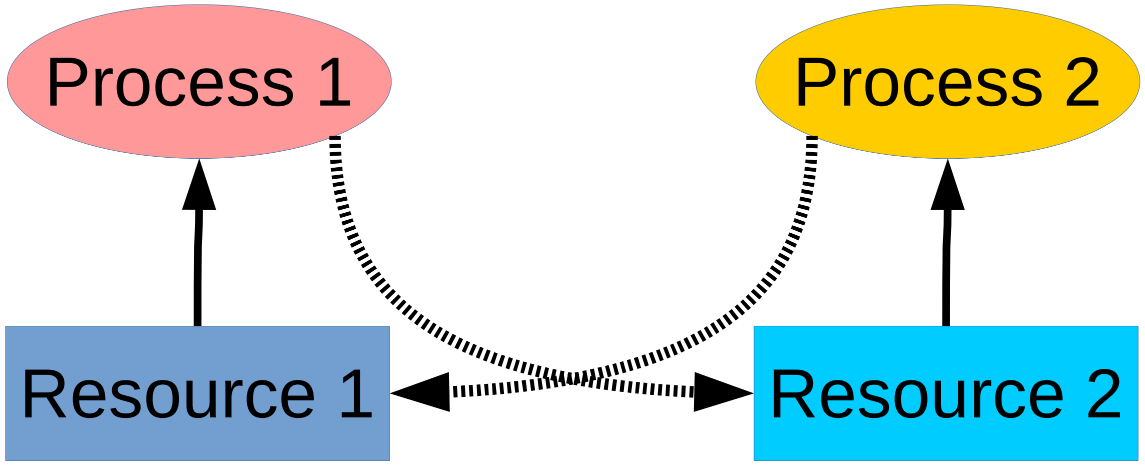
- **Deadlock as a reality**: Allow the deadlock of processes and resources to happen, be detected and for the system to recover
- **Ignore the deadlock:** Pretend that the deadlock is not happening. Hope for the best.

- **Wait: *Ignoring* deadlocks?**
  - **Is this really such a great idea??**
    - **Why or why not?**

Conditions for Resource DeadLocks

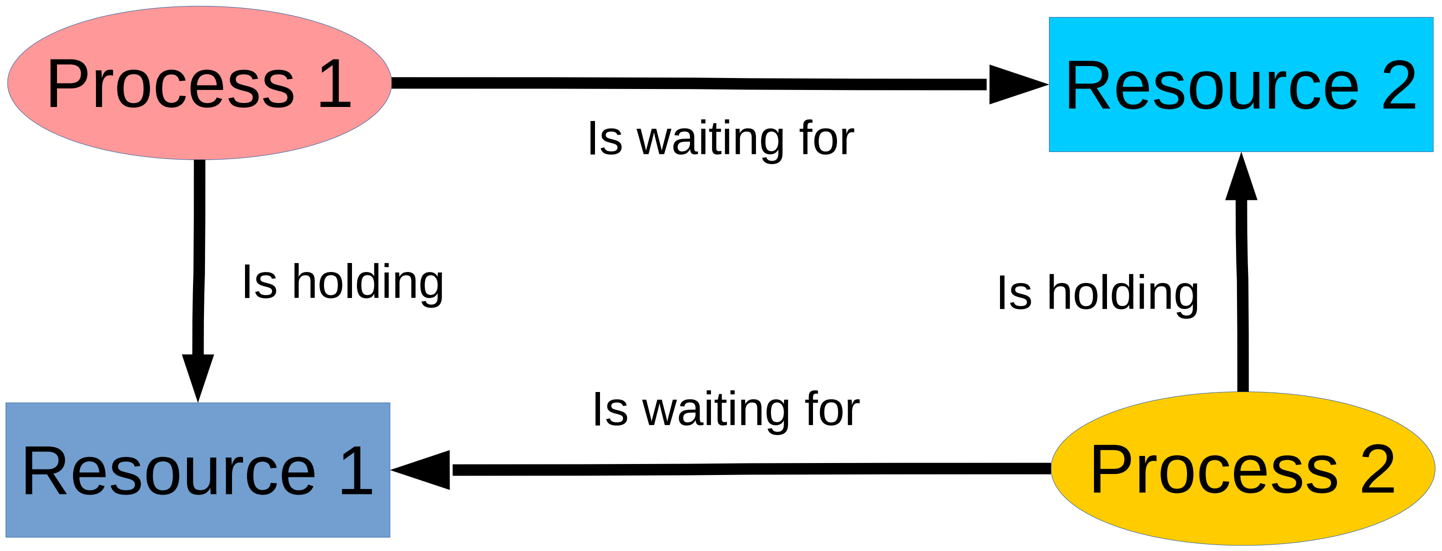
- By ensuring that at least one of the four conditions below do not hold, we can prevent deadlock.
- What are the conditions?
  - *Mutual exclusion condition*
  - *Hold and wait condition*
  - *No-preemption condition*
  - *Circular wait condition*

Mutual Exclusion



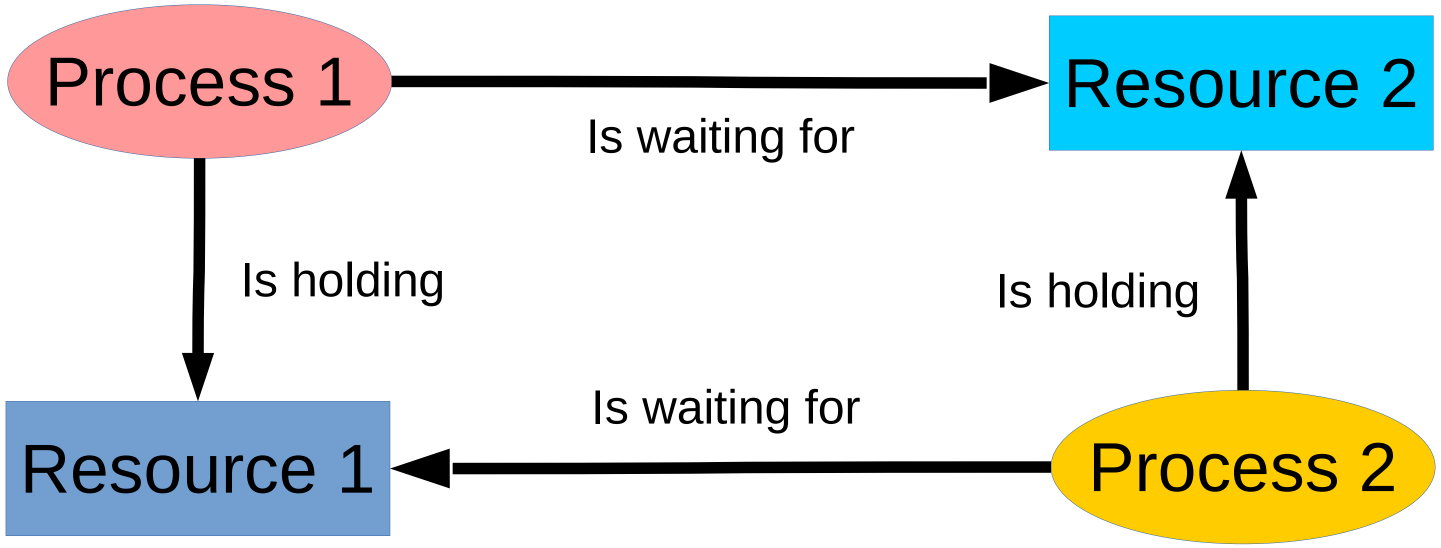
- Each resource is either currently assigned to exactly one process or is available
- Some sharable resources must be accessed exclusively (i.e, a printer) which means that we cannot deny the mutual exclusion condition.
- To avoid this condition: make sure that one process is able to give up a resource at all times

Hold and Wait (1)



- No process can hold some resource and then request for others
- Approach to avoid (1): Force a process to request *all* resources it needs at startup. The process dies, if not all resources are available.
- Approach to avoid (2): If a process needs to acquire a new resource, it must first release all resources it holds, then reacquire all it needs at one time.

Hold and Wait (2)



- Resource utilization may be low, since many resources will be held by a process and unused for a long time.
- Starvation: a process that needs some popular resource may have to wait indefinitely.

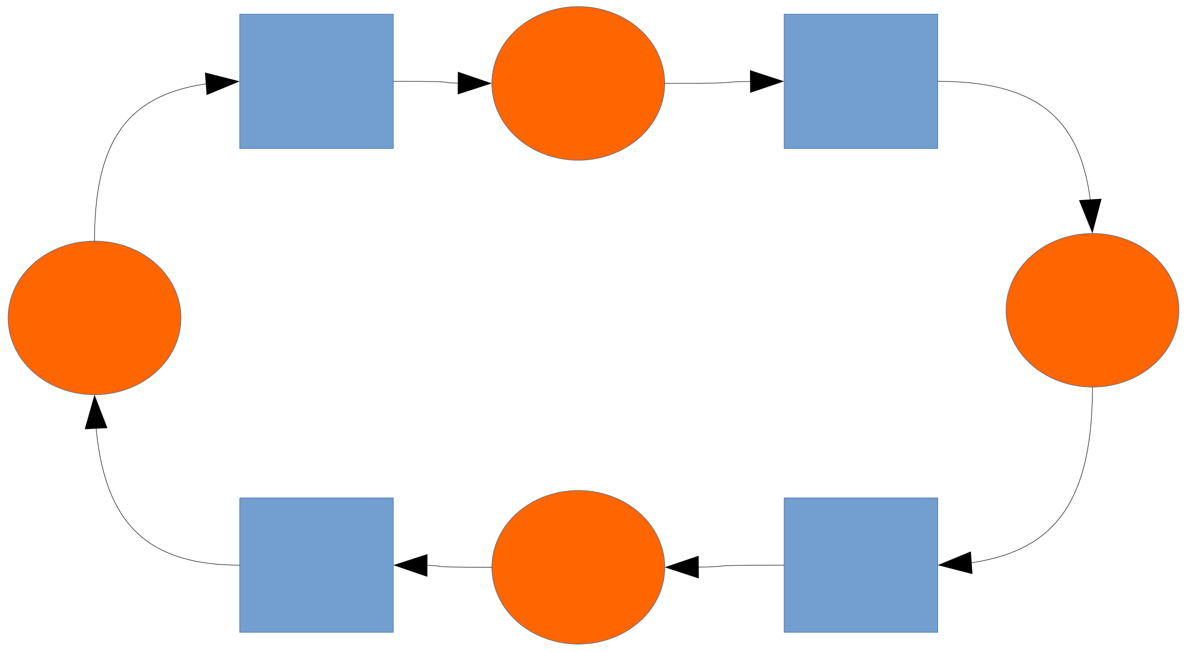
No-Preemption (1)

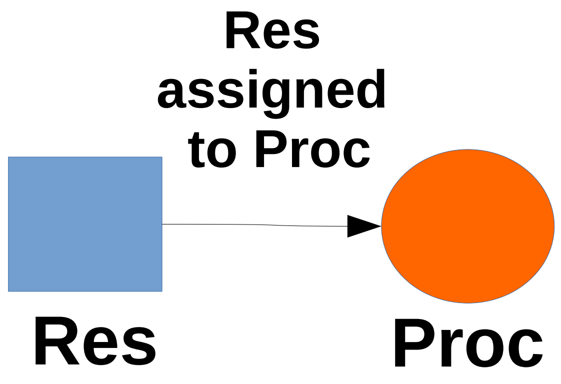
- *“If I cannot go now with all that I need, then I will give it all up and wait until all resources are free...”*
- Resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them.
- If a process is holding some resource and requesting others that are being held by other processes, the resources of the requesting process are preempted (released) for others to use them.

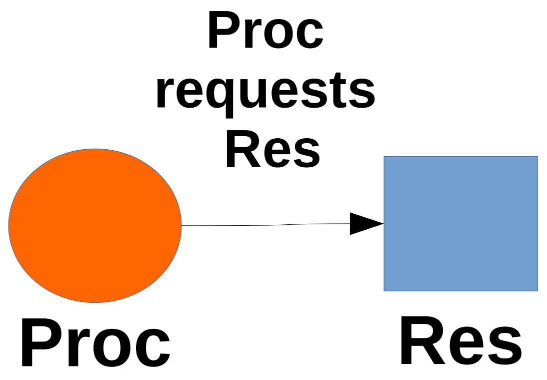
No-Preemption (2)

- If the requested resources are not available:
  - Resources are being held by processes that are waiting for additional resources, these resources are preempted and given to the requesting processes.
  - The requesting process waits until the requested resource is available. While waiting, its resources may be preempted.
  - This works only if the state of the process and the resources can be saved and restored easily (i.e, a CPU unit of time and memory)

Circular Wait

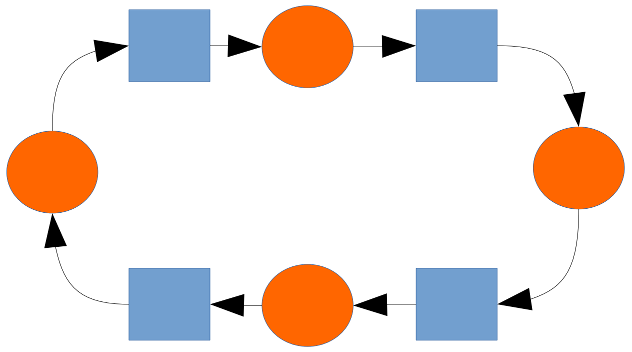






- Condition of *k* processes holding *k* resources
- There is a circular list of two or more processes, each of which is waiting for a resource held by the next member of the chain.

Circular wait Solutions



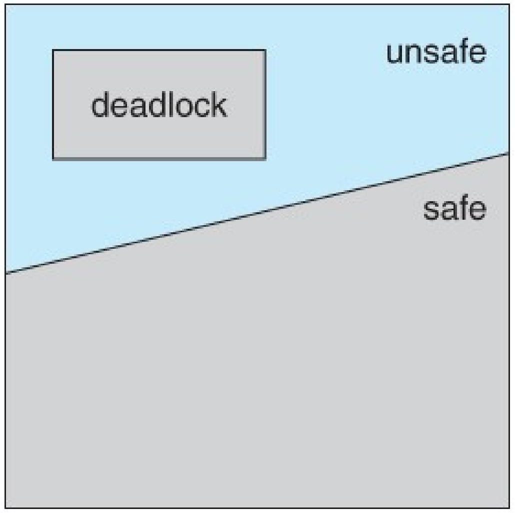
- There is a cycle in the graph of processes and resources
  - Choose a resource request strategy by which no such cycle can be introduced
- How? Resources are numbered and a process can only request a resource higher than the resource type it holds.
- Meaning: ***no*** process can request a resource ***lower*** than what it is currently holding.

Circular Wait Solution Example

- List of resources: tape drives, disks, and printers.
- Assign orderings:
  - Tapes → 1
  - Disks → 4
  - Printers → 8
- If a process holds a disk (4), then it can only ask for a printer (8) and cannot request a tape (1).
- A process must release some lower order resources to request another lower order resource.
  - To get tapes (1), a process must release a disk (4).

Deadlock States

Safe, unsafe, and deadlocked state spaces.

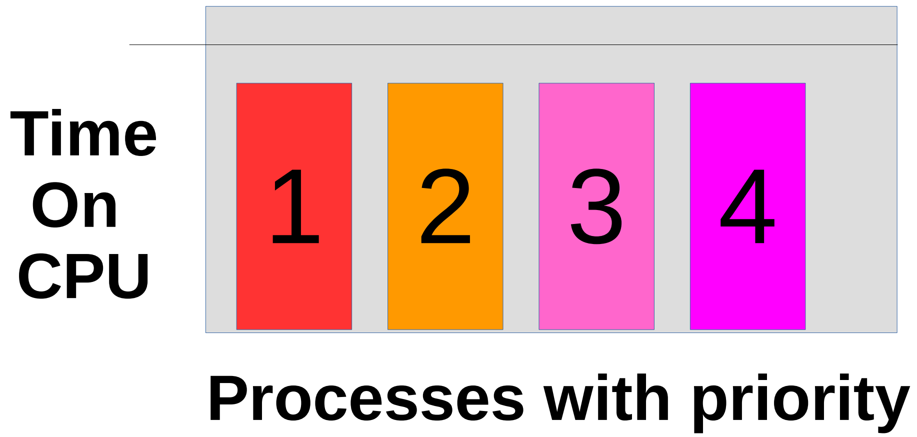


- A state is safe if the system can allocate all resources requested by all processes (up to their stated maximums) without entering a deadlock state.
- More formally, a state is safe if there exists a safe sequence of processes { P_0, P_1, P_2, ..., P_N } such that all of the resource requests for P_i can be granted using the resources currently allocated to P_i and all processes P_j where $j < i$. (i.e. if all the processes prior to P_i finish and free up their resources, then P_i will be able to finish also, using the resources that they have freed up.)
- If a safe sequence does not exist, then the system is in an unsafe state, which MAY lead to deadlock. (All safe states are deadlock free, but not all unsafe states lead to deadlocks.)

Avoiding Deadlock States

- Each process provides (to the OS) the maximum number of resources (of each type) that it will require to run to completion.
- The OS must now choose how to run processes to prevent (avoid) potential deadlocks.
- ***Avoidance***: Using information about resource requirements, scheduling algorithms can choose processes to ensure that their resource usage will never cause a system to enter a deadlock state.

A Word on Schedulers



- **Round Robin**: all processes get an equal slice of CPU time to run.
- **How could a deadlock happen with RR scheduler and no other avoidance mechanisms?**
  - **What kinds of processes and resource requirements are best for this scheduler?**
- **Take a minute to discuss with others.**