

Chapter VI

DEADLOCKS

(short version)

Jehan-François Pâris
jfp@uh.edu



Overview

- ***Deadlocks***
- ***Necessary conditions for deadlocks***
- ***Deadlock prevention***



Deadlocks

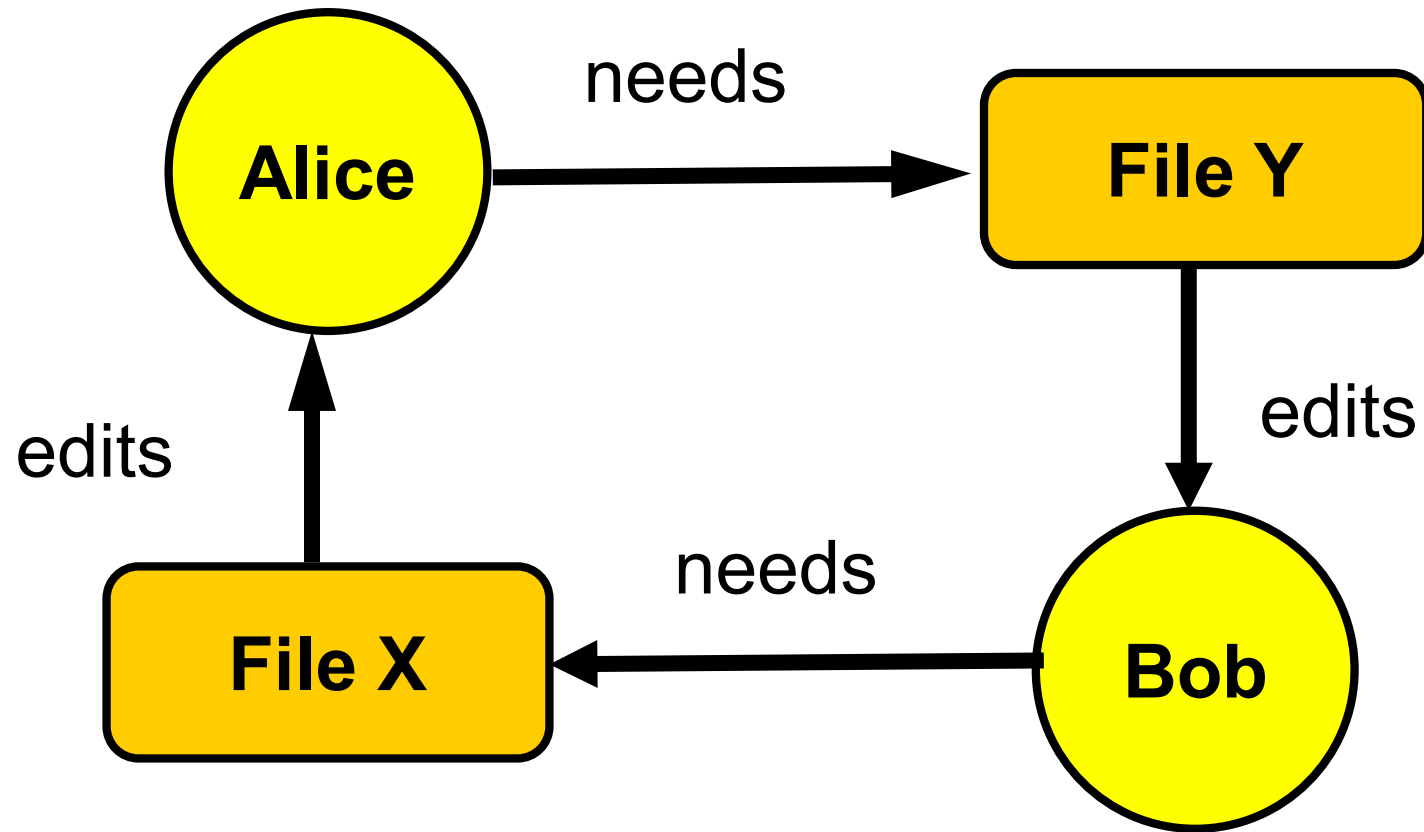
- A **deadlock** is said to occur whenever
 - Two or more processes are blocked
 - Each of these processes is waiting for a resource that is held by another blocked process.



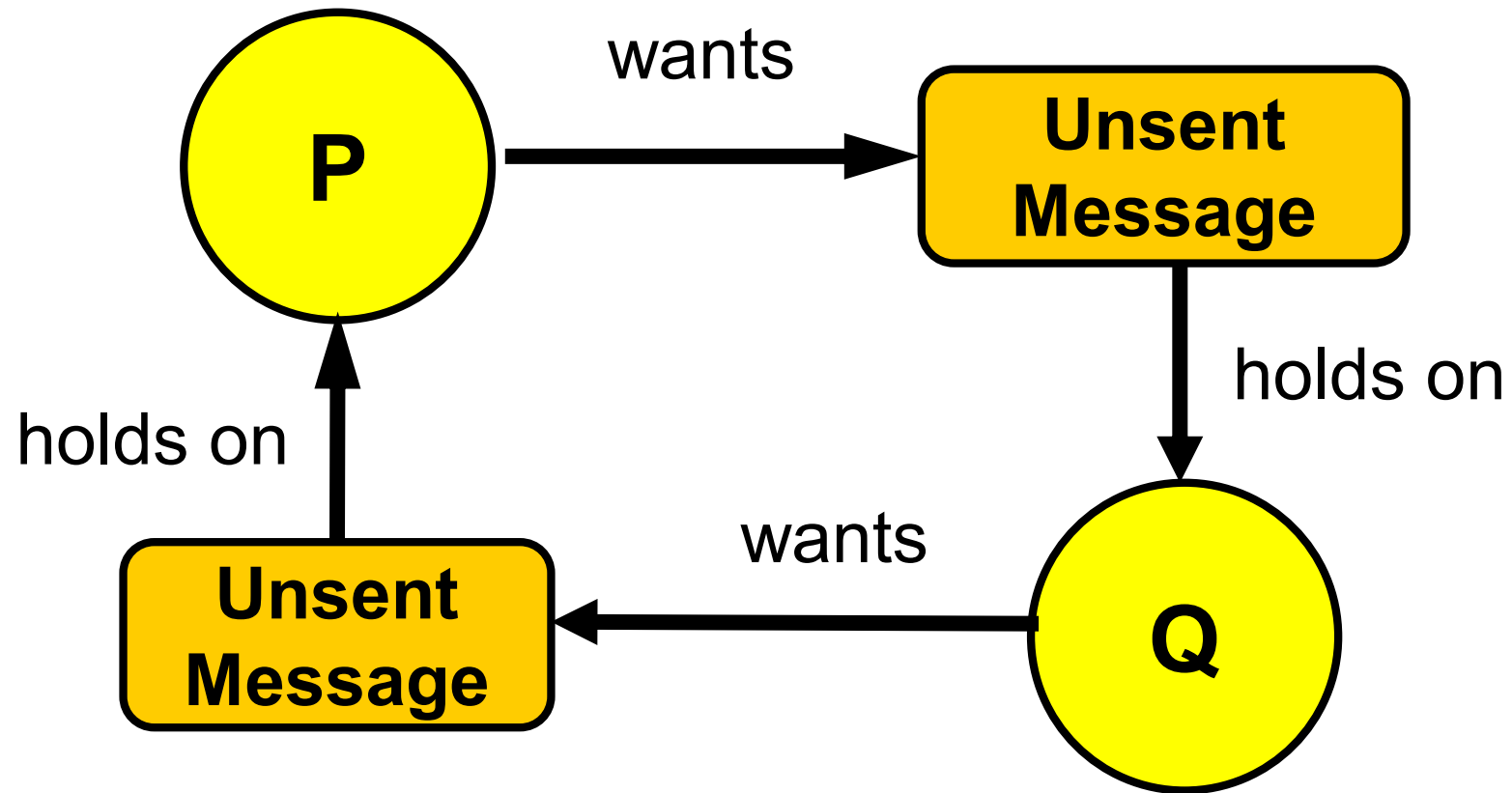
Examples

- Alice edits file X and needs to access file Y
 - Bob edits file Y and needs to access file X
- Process P expects a message from process Q
 - Process Q expects a message from process P

A graphic view (I)

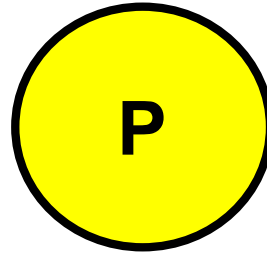


A graphic view (II)



Elements

■ Processes

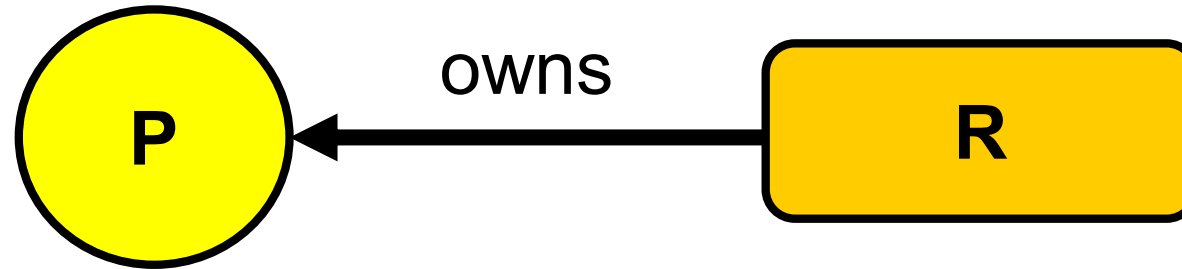


■ Resources

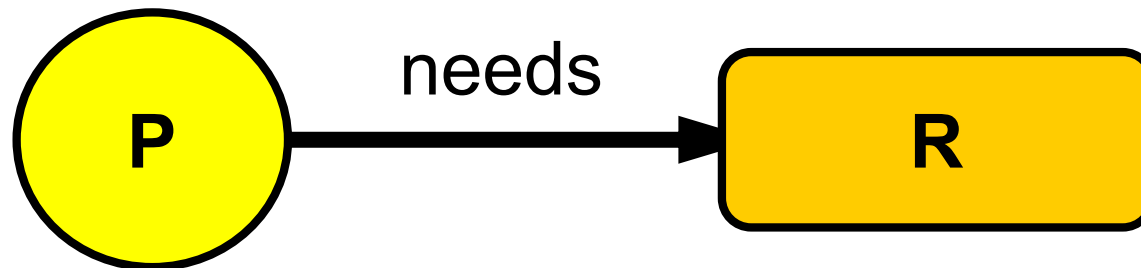


Relations

- Process P holds on/owns resource R



- Process P needs/wants resource R





Serially reusable resources

- Memory space, buffer space, disk space, USB slot to insert a flash drive
- Exist only in a *limited quantity*
- One process may have to *wait* for another process to release the resources it needs.



Consumable resources

- ***Cannot be reused***
- ***Messages*** are best example:
 - ***"Owned"*** by the process that creates them ***until it releases them***
 - ***"Wanted"*** by the process that waits for them

Handling deadlocks

- ***Do nothing:***
Ignore the problem
- ***Deadlock prevention:***
Build ***deadlock-free*** systems
- ***Deadlock avoidance:***
Avoid system states that *could* lead to a deadlock
- ***Deadlock detection:***
Detect and break deadlocks





Haberman's conditions

- Four ***necessary conditions*** must ***all*** be in effect for deadlocks to happen:
 - ***Mutual Exclusion***
 - ***Hold and Wait***
 - ***No Preemption***
 - ***Circular Wait***



Mutual exclusion

- At least one of the processes involved in the deadlock must claim ***exclusive control*** of some of the resources it requires
 - ***No sharing***



Hold and wait

- Processes can hold the resources that have already been allocated to them while waiting for additional resources



No preemption

- Once a resource has been allocated to a process, it ***cannot be taken away or borrowed*** from that process until the process is finished with it



Circular wait

- There must be a circular chain of processes such that each process in the chain holds some resources that are needed by the next process in the chain.
 - *Formal equivalent to what we call a **vicious circle***



Deadlock prevention

- ***Any*** system that prevents ***any*** of the four necessary conditions for deadlocks will be deadlock-free
- Must find the easiest condition to deny



Denying mutual exclusion

- Prevent any process from claiming ***exclusive control*** of any the resource
- **Drawbacks**
 - Many resources can only be used by one process at a time
 - Cannot hold on a message and send it at the same time



Denying hold and wait

- Require processes to get ***all*** the resources they will need or ***none of them***
- **Drawbacks**
 - Forces processes to acquire ahead of time all the resources they might need
 - Does not apply the consumable resources such a messages



Allowing preemption

- Let processes ***take away or borrow*** the resources they need from the processes that hold on them
- **Drawbacks**
 - Will result in ***lost work*** when a process steals storage space from another process
 - Cannot force processes to send messages



Denying circular wait (I)

- Impose a ***total order*** on all resource types and force all processes to follow that order when they acquire new resources
- If a process needs more than one unit of a given resource type it should acquire all of them or none



Denying circular wait (II)

- Works very well for resources like CPU and memory
- **Drawbacks**
 - Would force messages to move in only one direction
 - *Processes could not exchange messages*



A little problem

- Two courses at South Hillcroft University are ***co-requisites*** of each other:
 - *Sandcastle Design*
 - *History of Sandcastles*
- Both courses are likely to be oversubscribed
- Think of possible deadlocks
- Post your solutions on Prulu



Check list

- You must understand
 - Difference between consumable and serially-reusable resources
 - Haberman's four necessary conditions for deadlocks
 - The four ways to deny them
 - Why they do not work for client/server systems