Scheduling

Scheduling the CPU is clearly a difficult problem

Scheduling

Scheduling the CPU is clearly a difficult problem

It requires the collection and manipulation of many statistics about processes

Scheduling

Scheduling the CPU is clearly a difficult problem

It requires the collection and manipulation of many statistics about processes

Scheduling one resource (the CPU) is hard enough

Scheduling

Scheduling the CPU is clearly a difficult problem

It requires the collection and manipulation of many statistics about processes

Scheduling one resource (the CPU) is hard enough

We now look at a new problem that arises when we want to schedule *multiple* resources

Deadlock

Processes compete for resources like disks and network and the OS mediates this

Deadlock

Processes compete for resources like disks and network and the OS mediates this

To read from a disk, a process must call the OS kernel and wait for the kernel to reply

Terminology

When we say “a process waits for the kernel” we mean, of course, something entirely different

Terminology

When we say “a process waits for the kernel” we mean, of course, something entirely different

What actually happens is the kernel marks the process as blocked, and does not consider it for scheduling until the requested resource has arrived

Terminology

When we say “a process waits for the kernel” we mean, of course, something entirely different

What actually happens is the kernel marks the process as blocked, and does not consider it for scheduling until the requested resource has arrived

There is no “waiting” happening: the process does not run when blocked

Deadlock

Processes compete for resources like disks and network and the OS mediates this

To read from a disk, a process must call the OS kernel and wait for the kernel to reply

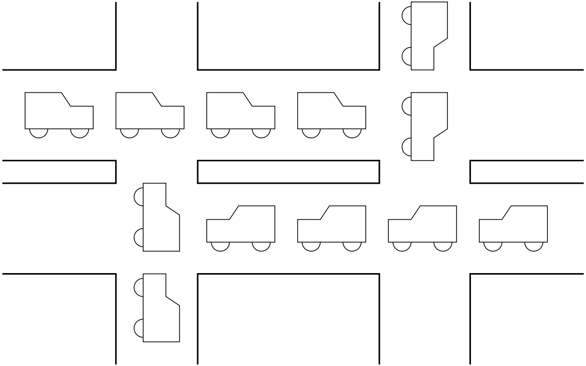
Deadlock

Processes compete for resources like disks and network and the OS mediates this

To read from a disk, a process must call the OS kernel and wait for the kernel to reply

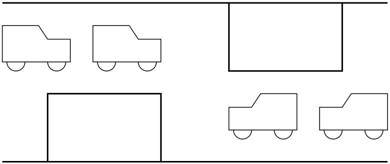
Sometimes the delay is infinite!

Deadlock



Gridlock/Deadlock

Deadlock



Gridlock/Deadlock

Deadlock

This can happen in an OS

Deadlock

This can happen in an OS

Process P_1 wants to copy some data from disk D_1 to disk D_2, while process P_2 wants to copy some data from disk D_2 to disk D_1

Deadlock

This can happen in an OS

Process P_1 wants to copy some data from disk D_1 to disk D_2, while process P_2 wants to copy some data from disk D_2 to disk D_1

- Initially P_1 is running and makes a request for access to D_2

Deadlock

This can happen in an OS

Process P_1 wants to copy some data from disk D_1 to disk D_2, while process P_2 wants to copy some data from disk D_2 to disk D_1

- Initially P_1 is running and makes a request for access to D_2
- The OS takes over and gives P_1 exclusive access to D_2 (not a smart OS)

Deadlock

This can happen in an OS

Process P_1 wants to copy some data from disk D_1 to disk D_2, while process P_2 wants to copy some data from disk D_2 to disk D_1

- Initially P_1 is running and makes a request for access to D_2
- The OS takes over and gives P_1 exclusive access to D_2 (not a smart OS)
- The OS decides to run P_2

Deadlock

This can happen in an OS

Process P_1 wants to copy some data from disk D_1 to disk D_2, while process P_2 wants to copy some data from disk D_2 to disk D_1

- Initially P_1 is running and makes a request for access to D_2
- The OS takes over and gives P_1 exclusive access to D_2 (not a smart OS)
- The OS decides to run P_2
- P_2 runs and makes a request for access to D_1

Deadlock

This can happen in an OS

Process P_1 wants to copy some data from disk D_1 to disk D_2, while process P_2 wants to copy some data from disk D_2 to disk D_1

- Initially P_1 is running and makes a request for access to D_2
- The OS takes over and gives P_1 exclusive access to D_2 (not a smart OS)
- The OS decides to run P_2
- P_2 runs and makes a request for access to D_1
- The OS takes over and gives P_2 exclusive access to D_1

Deadlock

- The OS decides to run P_1

Deadlock

- The OS decides to run P_1
- P_1 runs and makes a request for access to D_1

Deadlock

- The OS decides to run P_1
- P_1 runs and makes a request for access to D_1
- The OS takes over and notices P_2 has locked D_1, so P_1 must wait until P_2 has finished with it; P_1 moves to state blocked

Deadlock

- The OS decides to run P_1
- P_1 runs and makes a request for access to D_1
- The OS takes over and notices P_2 has locked D_1, so P_1 must wait until P_2 has finished with it; P_1 moves to state blocked
- The OS decides to run P_2: it can't run P_1 as it is blocked

Deadlock

- The OS decides to run P_1
- P_1 runs and makes a request for access to D_1
- The OS takes over and notices P_2 has locked D_1, so P_1 must wait until P_2 has finished with it; P_1 moves to state blocked
- The OS decides to run P_2: it can't run P_1 as it is blocked
- P_2 runs and makes a request for access to D_2

Deadlock

- The OS decides to run P_1
- P_1 runs and makes a request for access to D_1
- The OS takes over and notices P_2 has locked D_1, so P_1 must wait until P_2 has finished with it; P_1 moves to state blocked
- The OS decides to run P_2: it can't run P_1 as it is blocked
- P_2 runs and makes a request for access to D_2
- The OS takes over and notices P_1 has locked D_2, so P_2 must wait until P_1 has finished with it; P_2 moves to state blocked

Deadlock

- The OS decides to run P_1
- P_1 runs and makes a request for access to D_1
- The OS takes over and notices P_2 has locked D_1, so P_1 must wait until P_2 has finished with it; P_1 moves to state blocked
- The OS decides to run P_2: it can't run P_1 as it is blocked
- P_2 runs and makes a request for access to D_2
- The OS takes over and notices P_1 has locked D_2, so P_2 must wait until P_1 has finished with it; P_2 moves to state blocked
- Now both P_1 and P_2 are blocked and the OS can't run either process!

Deadlock

P_1 can't run until D_1 is free, but D_1 won't be free until P_2 runs

P_2 can't run until D_2 is free, but D_2 won't be free until P_1 runs

Deadlock

P_1 can't run until D_1 is free, but D_1 won't be free until P_2 runs

P_2 can't run until D_2 is free, but D_2 won't be free until P_1 runs

This is called *deadlock*

Deadlock

P_1 can't run until D_1 is free, but D_1 won't be free until P_2 runs

P_2 can't run until D_2 is free, but D_2 won't be free until P_1 runs

This is called *deadlock*

Deadlock can happen on any kind of shared resources that require exclusive access

Deadlock

P_1 can't run until D_1 is free, but D_1 won't be free until P_2 runs

P_2 can't run until D_2 is free, but D_2 won't be free until P_1 runs

This is called *deadlock*

Deadlock can happen on any kind of shared resources that require exclusive access

And with more than two processes: think of three or more processes in a circle

Deadlock

A formal definition:

Deadlock

A formal definition:

A set of processes D is *deadlocked* if

1. each process P_i in D is blocked on some event e_i
2. event e_i can only be caused by some process in D

Deadlock

Note that you can only get deadlock if

Deadlock

Note that you can only get deadlock if

- there is more than one resource

Deadlock

Note that you can only get deadlock if

- there is more than one resource
- there is more than one process

□

Deadlock

Note that you can only get deadlock if

- there is more than one resource
- there is more than one process[1]

□

[1] It could technically happen with just one process, but that would be quite dumb programming to request for a resource you already have

Deadlock

Note that you can only get deadlock if

- there is more than one resource
- there is more than one process[1][2]

[1] It could technically happen with just one process, but that would be quite dumb programming to request for a resource you already have

[2] I've seen it happen