Deadlock

Detection and Breaking

So that leaves breaking the deadlock: as always there are lots of ways we can do this, none terribly satisfactory

Deadlock

Detection and Breaking

So that leaves breaking the deadlock: as always there are lots of ways we can do this, none terribly satisfactory

- Kill one or more or all of the deadlocked processes: a bit drastic, but sometimes the only solution. But which process? For example, out of memory “OOM killers” are tricky to get right

Deadlock

Detection and Breaking

So that leaves breaking the deadlock: as always there are lots of ways we can do this, none terribly satisfactory

- Kill one or more or all of the deadlocked processes: a bit drastic, but sometimes the only solution. But which process? For example, out of memory “OOM killers” are tricky to get right
- Preempt the blocking resources: better, if possible. If there are multiple resources causing the deadlock we have to choose which, as preempting just a few might free things up enough

Deadlock

Detection and Breaking

So that leaves breaking the deadlock: as always there are lots of ways we can do this, none terribly satisfactory

- Kill one or more or all of the deadlocked processes: a bit drastic, but sometimes the only solution. But which process? For example, out of memory “OOM killers” are tricky to get right
- Preempt the blocking resources: better, if possible. If there are multiple resources causing the deadlock we have to choose which, as preempting just a few might free things up enough
- Add resources: rarely possible

Deadlock

Detection and Breaking

Exercise. Think about how you might apply deadlock prevention or breaking to a) Dining Philosophers and b) the car deadlock scenarios

Deadlock

In real life, a popular approach is simply to ignore the possibility of deadlock happening

Deadlock

In real life, a popular approach is simply to ignore the possibility of deadlock happening

Sometimes called the *Ostrich Algorithm*

Deadlock

In real life, a popular approach is simply to ignore the possibility of deadlock happening

Sometimes called the *Ostrich Algorithm*

There is not entirely stupid, as it argues that the costs associated with prevention or detection are large, and if deadlocks are rare, then the cost of an occasional reboot of the machine is small in comparison

Deadlock

In real life, a popular approach is simply to ignore the possibility of deadlock happening

Sometimes called the *Ostrich Algorithm*

There is not entirely stupid, as it argues that the costs associated with prevention or detection are large, and if deadlocks are rare, then the cost of an occasional reboot of the machine is small in comparison

In a carefully written OS, you can eliminate many of the possible causes of deadlock, or, at least, reduce the chances of them happening

Deadlock

Some resources are preemptable, e.g., memory (as we shall discuss in depth later), but a more general solution (also applied to memory) is *virtualisation*, where the OS pretends each process has sole access to a resource

Deadlock

Some resources are preemptable, e.g., memory (as we shall discuss in depth later), but a more general solution (also applied to memory) is *virtualisation*, where the OS pretends each process has sole access to a resource

We have already seen this for printers in the form of *spooling*

Deadlock

Some resources are preemptable, e.g., memory (as we shall discuss in depth later), but a more general solution (also applied to memory) is *virtualisation*, where the OS pretends each process has sole access to a resource

We have already seen this for printers in the form of *spooling*

A process thinks it is writing to a printer, but it is actually writing to a tape, and the tape is later written to the printer

Deadlock

Similarly, for example, a process thinks it writes to a network card but the data is actually buffered by the OS somewhere in memory, to be sent later when the card is free

Deadlock

Similarly, for example, a process thinks it writes to a network card but the data is actually buffered by the OS somewhere in memory, to be sent later when the card is free

And so on for other kinds of devices: a process interfaces with its own virtualised device, there is no possibility of deadlock as every process can progress without waiting, and the OS sorts out transferring the data to or from the real device

Deadlock

Similarly, for example, a process thinks it writes to a network card but the data is actually buffered by the OS somewhere in memory, to be sent later when the card is free

And so on for other kinds of devices: a process interfaces with its own virtualised device, there is no possibility of deadlock as every process can progress without waiting, and the OS sorts out transferring the data to or from the real device

But, of course, this new perspective just shifts the actual problem: when and in what order should the OS do the I/O?

Deadlock

Similarly, for example, a process thinks it writes to a network card but the data is actually buffered by the OS somewhere in memory, to be sent later when the card is free

And so on for other kinds of devices: a process interfaces with its own virtualised device, there is no possibility of deadlock as every process can progress without waiting, and the OS sorts out transferring the data to or from the real device

But, of course, this new perspective just shifts the actual problem: when and in what order should the OS do the I/O?

This is called *I/O scheduling*

Deadlock

A printer could have simple first-in-first out queue, but other devices (disks, etc.) require something more sophisticated

Deadlock

A printer could have simple first-in-first out queue, but other devices (disks, etc.) require something more sophisticated

For example, a typical disk driver will re-order writes to a disk match the physical movements of the write head

Deadlock

A printer could have simple first-in-first out queue, but other devices (disks, etc.) require something more sophisticated

For example, a typical disk driver will re-order writes to a disk match the physical movements of the write head

This is a topic we won’t have time to go into!

Deadlock

Priority Inversion

One last word on deadlock, this one caused by process priorities and non-preemptible resources

Deadlock

Priority Inversion

One last word on deadlock, this one caused by process priorities and non-preemptible resources

Recall processes have priorities for scheduling purposes

Deadlock

Priority Inversion

One last word on deadlock, this one caused by process priorities and non-preemptible resources

Recall processes have priorities for scheduling purposes

- Suppose a low priority process L holds some resource

Deadlock

Priority Inversion

One last word on deadlock, this one caused by process priorities and non-preemptible resources

Recall processes have priorities for scheduling purposes

- Suppose a low priority process L holds some resource
- A high priority process H is scheduled

Deadlock

Priority Inversion

One last word on deadlock, this one caused by process priorities and non-preemptible resources

Recall processes have priorities for scheduling purposes

- Suppose a low priority process L holds some resource
- A high priority process H is scheduled
- H requests the resource

Deadlock

Priority Inversion

One last word on deadlock, this one caused by process priorities and non-preemptible resources

Recall processes have priorities for scheduling purposes

- Suppose a low priority process L holds some resource
- A high priority process H is scheduled
- H requests the resource
- It can’t get it as it is still held by L, so H is blocked

Deadlock

Priority Inversion

One last word on deadlock, this one caused by process priorities and non-preemptible resources

Recall processes have priorities for scheduling purposes

- Suppose a low priority process L holds some resource
- A high priority process H is scheduled
- H requests the resource
- It can’t get it as it is still held by L, so H is blocked
- Eventually, when L is done, H will be able to run

Deadlock

Priority Inversion

The low priority process is preventing the high priority process from running

Deadlock

Priority Inversion

The low priority process is preventing the high priority process from running

This is called *priority inversion*

Deadlock

Priority Inversion

The low priority process is preventing the high priority process from running

This is called *priority inversion*

What is worse, other processes M of intermediate priority (that don't need the resource) can preempt L, preventing it running, and thus make the time H has to wait indefinitely long

Deadlock

Priority Inversion

The low priority process is preventing the high priority process from running

This is called *priority inversion*

What is worse, other processes M of intermediate priority (that don't need the resource) can preempt L, preventing it running, and thus make the time H has to wait indefinitely long

If H is some real-time operation this can be serious

Deadlock

Priority Inversion

Fixes include

Deadlock

Priority Inversion

Fixes include

Priority inheritance The priority of H is temporarily loaned to L for the time it needs the resource. This ensures L can run and get out of the way

Deadlock

Priority Inversion

Priority ceilings Each *resource* is given a priority equal to the highest priority of any task that might want to grab that resource

Deadlock

Priority Inversion

Priority ceilings Each *resource* is given a priority equal to the highest priority of any task that might want to grab that resource

When L gets the resource its priority is temporarily boosted to the priority ceiling of the resource: either immediately, or when another process tries to grab the resource

Deadlock

Priority Inversion

Priority ceilings Each *resource* is given a priority equal to the highest priority of any task that might want to grab that resource

When L gets the resource its priority is temporarily boosted to the priority ceiling of the resource: either immediately, or when another process tries to grab the resource

No other process that would want to grab the resource can be scheduled

Deadlock
Priority Inversion

Priority ceilings Each *resource* is given a priority equal to the highest priority of any task that might want to grab that resource

When L gets the resource its priority is temporarily boosted to the priority ceiling of the resource: either immediately, or when another process tries to grab the resource

No other process that would want to grab the resource can be scheduled

Determining the ceiling is tricky, as it needs knowledge of the possible needs of processes

Deadlock

Priority Inversion

Disable scheduling preemption during use of non-preemptible resources. Only feasible if you keep the periods of use very short. Quite a popular solution for some resources, e.g., networks and disks, that are serviced very quickly

□

Deadlock

Priority Inversion

Disable scheduling preemption during use of non-preemptible resources. Only feasible if you keep the periods of use very short. Quite a popular solution for some resources, e.g., networks and disks, that are serviced very quickly

Exercise. Read up on these and other solutions

□