Deadlock

There are two approaches to the problem of deadlock

Deadlock

There are two approaches to the problem of deadlock

1. Prevention. Stopping it happening ever by preventing one of the conditions occurring

Deadlock

There are two approaches to the problem of deadlock

1. Prevention. Stopping it happening ever by preventing one of the conditions occurring
2. Detection and Breaking. Letting deadlock happen, but spotting when it does and then breaking it by destroying one of the conditions

Deadlock

Prevention can be further refined

Deadlock

Prevention can be further refined

1a. Prevention. Constrain resource allocation to prevent at least one of the four conditions (e.g., ensure hold-and-wait never happens)

Deadlock

Prevention can be further refined

1a. Prevention. Constrain resource allocation to prevent at least one of the four conditions (e.g., ensure hold-and-wait never happens)

1b. Avoidance. Be careful not to allocate a resource if it can be determined that it might possibly lead to a deadlock in the future: keeping the system in a “safe” state

Deadlock

Prevention can be further refined

1a. Prevention. Constrain resource allocation to prevent at least one of the four conditions (e.g., ensure hold-and-wait never happens)

1b. Avoidance. Be careful not to allocate a resource if it can be determined that it might possibly lead to a deadlock in the future: keeping the system in a “safe” state

Avoidance is harder to manage as it needs to predict future requests for resources, but tends to be more efficient as it can allocate resources that prevention would disallow

Deadlock

Prevention

We can prevent deadlocks by disallowing any of the conditions

Deadlock

Prevention

Breaking Mutual Exclusion

Deadlock

Prevention

Breaking Mutual Exclusion

This, quite often, cannot be broken

Deadlock

Prevention

Breaking Mutual Exclusion

This, quite often, cannot be broken

For example, trying to read a disk at the same time as another process is writing to it is a physical impossibility

Deadlock

Prevention

Breaking Mutual Exclusion

This, quite often, cannot be broken

For example, trying to read a disk at the same time as another process is writing to it is a physical impossibility

A lot of hardware only works if there is exclusive access, e.g., printers, sound cards, etc.

Deadlock

Prevention

Breaking Mutual Exclusion

This, quite often, cannot be broken

For example, trying to read a disk at the same time as another process is writing to it is a physical impossibility

A lot of hardware only works if there is exclusive access, e.g., printers, sound cards, etc.

Therefore we should take care to not hold on to such a resource for longer than is absolutely necessary

Deadlock

Prevention

Breaking Hold-and-wait

Deadlock

Prevention

Breaking Hold-and-wait

We can require a process not to hold any resources if it ever gets blocked on another resource

Deadlock

Prevention

Breaking Hold-and-wait

We can require a process not to hold any resources if it ever gets blocked on another resource

This has the non-progress feature, as noted previously, and can be very inefficient with much grabbing and releasing to no avail

Deadlock

Prevention

Breaking Hold-and-wait

We might require a process to request all necessary resources simultaneously, blocking until all are available

Deadlock

Prevention

Breaking Hold-and-wait

We might require a process to request all necessary resources simultaneously, blocking until all are available

- This might prevent the process from doing useful other work while one of the resources is unavailable but not yet needed by the process

Deadlock

Prevention

Breaking Hold-and-wait

We might require a process to request all necessary resources simultaneously, blocking until all are available

- This might prevent the process from doing useful other work while one of the resources is unavailable but not yet needed by the process
- Resources given to a process might be only needed much later, denying them to other processes in the meantime

Deadlock

Prevention

Breaking Hold-and-wait

We might require a process to request all necessary resources simultaneously, blocking until all are available

- This might prevent the process from doing useful other work while one of the resources is unavailable but not yet needed by the process
- Resources given to a process might be only needed much later, denying them to other processes in the meantime
- It may be that a process does not even know what resources it might need in advance, so this can be impossible to do anyway

Deadlock

Prevention

Breaking Hold-and-wait

A variant of this is not even to admit a process until all resources are available: this is even worse

Deadlock

Prevention

Breaking Hold-and-wait

A variant of this is not even to admit a process until all resources are available: this is even worse

Perhaps a process only needs to write to disk at the end of a 2 hour compute session: do we really want to lock the disk for 2 hours?

Deadlock

Prevention

Breaking No Preemption

Deadlock

Prevention

Breaking No Preemption

This may only be possible for certain kinds of resource, namely those whose state can easily be saved and restored

Deadlock

Prevention

Breaking No Preemption

This may only be possible for certain kinds of resource, namely those whose state can easily be saved and restored

The OS might choose to preempt the holding process and take the resource away from it, giving it back later when the process is scheduled again

Deadlock

Prevention

Breaking No Preemption

This may only be possible for certain kinds of resource, namely those whose state can easily be saved and restored

The OS might choose to preempt the holding process and take the resource away from it, giving it back later when the process is scheduled again

This would be confusing for the holding process as the resource might change while it was owned by another process

Deadlock

Prevention

Thus, the resource should be given back to the process in an equivalent state to it was in when it was preempted, so the process can continue from where it left off

Deadlock

Prevention

Thus, the resource should be given back to the process in an equivalent state to it was in when it was preempted, so the process can continue from where it left off

For some resources this is possible, e.g., memory

Deadlock

Prevention

Thus, the resource should be given back to the process in an equivalent state to it was in when it was preempted, so the process can continue from where it left off

For some resources this is possible, e.g., memory

For others, not. For example, a printer

Deadlock

Prevention

Breaking Circular Waits

Deadlock

Prevention

Breaking Circular Waits

One possible solution is to put an ordering on resources

$$R_1 < R_2 < R_3 < \ldots$$

Deadlock

Prevention

Breaking Circular Waits

One possible solution is to put an ordering on resources

$$R_1 < R_2 < R_3 < \ldots$$

E.g., (much simplified)

$$\text{disk } 1 < \text{disk } 2 < \text{printer} < \ldots$$

Deadlock

Prevention

Then:

> *A process that holds resource R may then only request resources that are after R in the order*

Deadlock

Prevention

Then:

> *A process that holds resource R may then only request resources that are after R in the order*

In our example, if you have grabbed the printer, you cannot grab a disk

Deadlock

Prevention

Then:

> *A process that holds resource R may then only request resources that are after R in the order*

In our example, if you have grabbed the printer, you cannot grab a disk

If a process makes such a request, the OS simply refuses to grant it

Deadlock

Prevention

Then:

> *A process that holds resource R may then only request resources that are after R in the order*

In our example, if you have grabbed the printer, you cannot grab a disk

If a process makes such a request, the OS simply refuses to grant it

The process might choose to drop the printer and re-request the disk

Deadlock

Prevention

Breaking Circular Waits

Now we cannot deadlock, as a deadlock would imply A has grabbed R_i and requested R_j; while B has grabbed R_j and requested R_i

Deadlock

Prevention

Breaking Circular Waits

Now we cannot deadlock, as a deadlock would imply A has grabbed R_i and requested R_j; while B has grabbed R_j and requested R_i

For this to happen we would have both

$$i < j \qquad \text{and} \qquad j < i$$

and this is impossible

Deadlock

Prevention

Breaking Circular Waits

This suffers the same problems as Hold-and-wait, namely inefficiency and unnecessarily holding resources

Deadlock

Prevention

Breaking Circular Waits

This suffers the same problems as Hold-and-wait, namely inefficiency and unnecessarily holding resources

Further, it works only if the process can make requests in increasing order; not always possible as it is not always possible to know what you need in advance

Deadlock

Prevention

Breaking Circular Waits

This suffers the same problems as Hold-and-wait, namely inefficiency and unnecessarily holding resources

Further, it works only if the process can make requests in increasing order; not always possible as it is not always possible to know what you need in advance

And if you have R_1 and R_3, but then want R_2 you have to drop R_3, get R_2, then regain R_3; very inefficient

Deadlock

Prevention

Breaking Circular Waits

This suffers the same problems as Hold-and-wait, namely inefficiency and unnecessarily holding resources

Further, it works only if the process can make requests in increasing order; not always possible as it is not always possible to know what you need in advance

And if you have R_1 and R_3, but then want R_2 you have to drop R_3, get R_2, then regain R_3; very inefficient

This usually effectively reduces to the request-all-at-once scenario