Deadlock – The Sequel

Graph Theoretical Treatment
Cycles and Knots
Detection, Prevention, and Avoidance
The Bankers Algorithm

Deadlock Detection

Is it or is it not a deadlock – this is the question.....

The determination of whether or not as system of process is deadlocked, is the task of Deadlock Detection!

Graph Reduction → reducing the resource graph using the graph reduction algorithm.

We need to recall some basic definitions from graph theory!

Some Definitions:

- G(V, E) → a graph with vertex set V and edge set E.
- V = {a, b, …, z}
- E = {(a,b), .. (z, b)}
- edge (z, b) is directed from z to b
- Recall → a resource graph G(V,E) is a digraph

…more definitions

- node z is a sink, if there are no edges (z, b).
- z is an isolated node if there are no edges (z, b) or (b, z).
- A path is a sequence (a,b,c ..y,z) of at least two nodes for which (a, b) ε E.
- A cycle is a path with the same first and last vertex.
- The reachable set of a node z, reachable(z) = {b | there is a path from z to b}
- A knot, K, is a non-empty set of nodes with the property that for all nodes z in K, reachable(z) = K.

Some notable facts:

1. in a digraph, knot → cycle
2. no node in a knot is a sink
3. there is no path from a node in a knot to a sink.
4. a digraph is free of knots if and only if, for each node z, z is a sink or there is a path from z to a sink.

What does "if and only if" mean?

…from the textbook

At this point, we will consider a somewhat constraint version of the Resource Graph Reduction Algorithm presented in the TB.

Given a resource graph G, repeat the following steps until there are no more blocked processes remaining:

1. select an unblocked process *p*
2. Remove *p*, including all request and allocation edges.

A resource graph is completely reducible if, at termination of the reduction sequence (steps 1 & 2), all process nodes have been deleted.

The system state represented by G is a deadlock state if G is not completely reducible.

ACCORDING TO THE TEXTBOOK:

→all reduction sequences of a given resource graph lead to the same final graph; i.e., it does not matter which unblocked process is selected in step 1 of the algorithm.

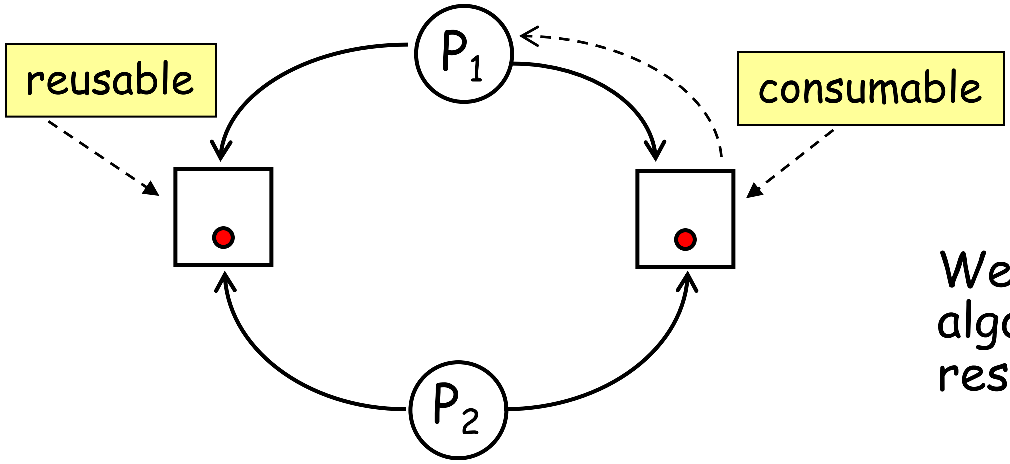
IS THIS TRUE IN GENERAL??

… in general

Consider the situation where both, reusable and consumable resources are involved.

The Situation depicted below:

1. P1 requests 1 unit of R1 and R2
2. P1 produces R1
3. P2 requests 1 unit of R1 and R2



So….. what if we reduce G by process P2 first?

What if G is reduced by P1 first??

→ Reducibility may be order dependent!!

Is our textbook wrong ??

Of course not – it just neglects to consider the more general case when consumable resources are involved.

We need to refine our reduction algorithm to take consumable resources into consideration.

General Graph Reduction

Formally, a graph may be reduced by a non-isolated node, representing an unblocked process P_i, using the following steps:

1. For each resource R_j, delete all edges (P_i,R_j) and if R_j is consumable, decrement r_j (the number of resource units) by the number of deleted request edges.

2. For each resource R_j, delete all edges (R_j,P_i). If R_j is reusable, then increment r_j by the number of deleted edges. If R_j is consumable (and P_i is the producer), set r_j to infinity.

Theorem:

A process P_i is not deadlocked in state S if and only if (iff) there exists a sequence of reductions in the corresponding graph G that leaves P_i unblocked.

Corollary:

If a graph G is completely reducible, then the state it represents is not deadlocked.

However, the converse of this corollary is NOT true!

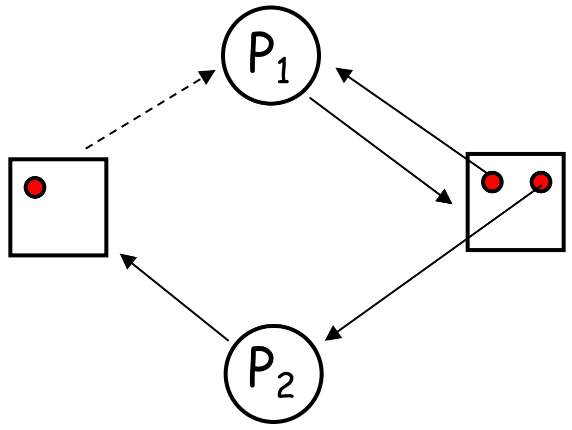
Expediency

One resource allocation policy that requires that all satisfiable resource request are granted immediately.

→in such system, the resource graph will never contain any staisfiable request edge.

→This is referred to as an EXPEDIENT system or systems state.

→In an expedient system, all processes that are requesting resources are blocked.



A non-expedient General Resource Graph

P_2 is requesting a resource but

P_2 is not blocked. R is available!

cycle and knots.. oh my!

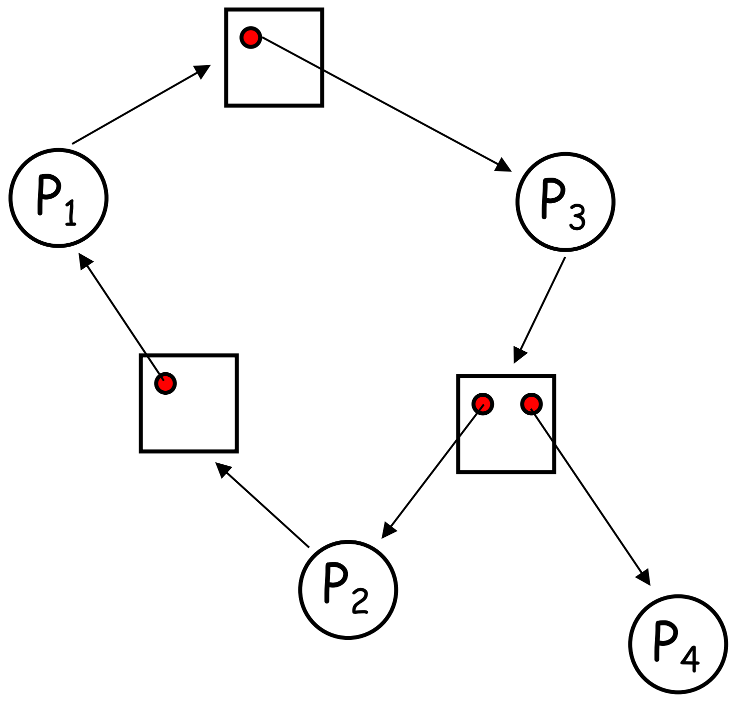
Theorem:

A cycle is a necessary condition for deadlock. If the graph is expedient, a knot is a sufficient condition for deadlock!

However, a knot is not a necessary condition for deadlock!
Why not ??

In summary:

- deadlock ➔ cycle
- cycle !➔ deadlock
- expedient & knot ➔ deadlock
- deadlock !➔ knot



Cycle but….. NO DEADLOCK!

Special Scenarios…

Single-Unit Resource

In many situation we deal with resource classes that consist of only one unit of the resource.

Here, a cycle is a sufficient condition for deadlock, leading to an efficient deadlock detection mechanism.

→It suffices to inspect the graph for cycles. How do we do this ??

Since each resource has at most one resource allocation edge attached to it, we can simplify the general resource graph to a wait-for-graph.

In the wait-for-graph, edges are drawn between processes. An edge (p_i, p_j) indicated that process p_i is waiting for a resource that is currently held by process p_j.

Here, cycle ➔ deadlock !! WHY?

Can you prove this?

…more scenarios

Single-Unit requests

In these systems a process may have at most one outstanding request for a unit of some resource.

What is different when compared to the Single-Unit Resource scenario?

→We may in fact have multiple units of a particular resource available, however, a process may not request more than one unit!!

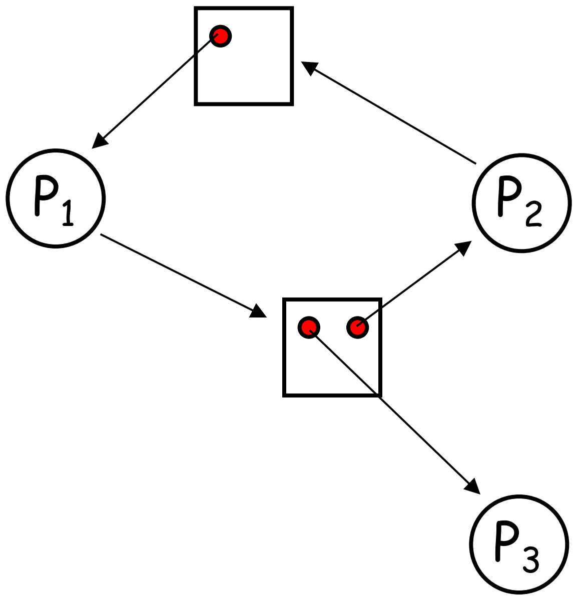
→So what …. How does this make things easier?

A single-unit request system is also expedient, hence knot ➔ deadlock !

Observations (Single-Unit Request):

1. if a graph is a deadlock state, then a knot exists.
2. a knot is both necessary and sufficient for deadlock to occur.
3. two reduction sequences lead to the same irreducible state
4. process P is not deadlocked in state S iff P is a sink or on a path to a sink.

…single-unit request example



- cycle but no deadlock
- expedient systems
- single-unit request

- It suffices to detect the existence of a knot in order to detect deadlock.

- How can you prove that the absence of a knot implies that the graph is reducible ?
  - If we assume that no knot exists, we can show that there is a sequence by which the graph can be reduced.
  - There must be a sink, and this sink-node must be a process because of expedience.

Reusable Resources Only

This is a special case of the general resource graph, which is considered in the textbook.

As discussed earlier, this IS the basis for efficient deadlock detection.

Theorem:

Let S be any state of a reusable resource system. Any sequence of reductions of the corresponding graph leads to the same unique graph that cannot be reduced further. S is not a deadlock state iff S is completely reducible.

How can we prove this theorem?

- observe that a reduction never decreases the number of available resources.
- hence, reducibility is NOT order dependent.
- Consequently, the reduction must end in the same unique state; either completely reduced or deadlocked.