Comp 204: Computer Systems and Their Implementation

Lecture 9: Deadlock

Today

- Deadlock
  - Definition
  - Resource allocation graphs
  - Detecting and dealing with deadlock

Deadlock

"When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone."

-- Kansas law

- A set of processes is deadlocked (in **deadly embrace**) if each process in the set is waiting for an event only another process in the set can cause.
- These events usually relate to resource allocation

Resource Allocation

- OS must allocate and share resources sensibly
- Resources may be
  - CPUs
  - Peripheral devices (printers etc.)
  - Memory
  - Files
  - Data
  - Programming objects such as semaphores, object locks etc.
- Usual process/thread sequence is request-use-release
  - Often via system calls

Creating Deadlock

- In its simplest form, deadlock will occur in the following situation:
  - process A is granted resource X
    and then requests resource Y
  - process B is granted resource Y
    and then requests resource X
  - both resources are non-shareable
    (e.g. tape drive, printer)
  - both resources are non-preemptible
    (i.e. cannot be taken away from their owner processes)

Question

- Consider the following situation regarding two processes (A and B), and two resources (X and Y):
  - Process A is granted resource X and then requests resource Y.
  - Process B is granted resource Y and then requests resource X.

- Which of the following is (are) true about the potential for deadlock?

I. Deadlock can be avoided by sharing resource Y between the two processes
II. Deadlock can be avoided by taking resource X away from process A
III. Deadlock can be avoided by process B voluntarily giving up its control of resource Y

a) I only
b) I and II only
c) I and III only
d) II and III only
e) I, II and III

Answer: e
I, II and III – as all three options will avoid exclusive ownership of the resources.

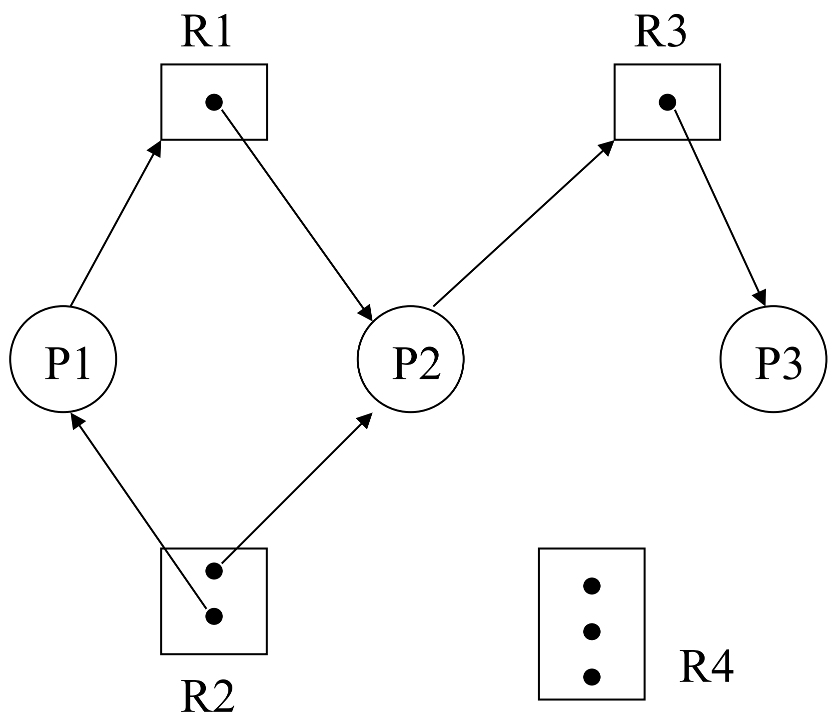
Resource Allocation Graphs

- Consist of a set of vertices *V* and a set of edges *E*
  - *V* is partitioned into two types:
    - Set of processes, $P = \{P_1, P_2, \ldots, P_n\}$
    - Set of resource types, $R = \{R_1, R_2, \ldots, R_m\}$
      - e.g. printers
      - Include instances of each type

Resource Allocation Graphs

- *E* is a set of directed edges
  - Request edge – from process to resource type, denoted $P_i \rightarrow R_j$
    - States that process P_i has requested an instance of resource type R_j and is currently waiting for it
  - Assignment edge – from resource instance to process, denoted $R_j \rightarrow P_i$
    - States that an instance of a resource type R_j has been allocated to process P_i
  - Request edges are transformed to assignment edges when request satisfied

Example Graph



No cycles, so no deadlock.

Example Graph

- The previous diagram depicts the following:

- Processes, resource types, edges
  - $P = \{P_1, P_2, P_3\}$
  - $R = \{R_1, R_2, R_3, R_4\}$
  - $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$

- Resource instances:
  - One instance of resource type R_1
  - Two instances of resource type R_2
  - One instance of resource type R_3
  - Three instances of resource type R_4

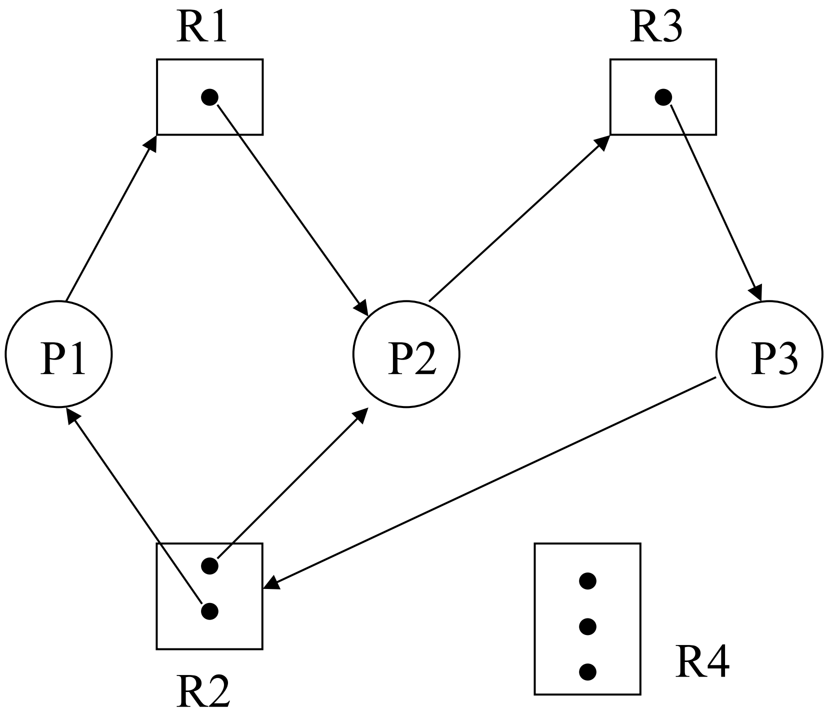
Example Graph

- Process states:
  - Process P_1 is holding an instance of resource type R_2 and is waiting for an instance of resource type R_1
  - Process P_2 is holding an instance of resource type R_1 and R_2 and is waiting for an instance of resource type R_3
  - Process P_3 is holding an instance of resource type R_3

Resource Allocation Graphs

- In resource allocation graphs we can show that deadlock has not occurred if there are no cycles in the graph
- If cycles do exist in the graph, this indicates that deadlock *may* be present
  - If each resource type consists of exactly one instance, a cycle indicates that deadlock has occurred
  - If each resource type consists of several instances, a cycle does not necessarily indicate that deadlock has occurred
- Example: On previous graph, suppose P_3 now requests R_2…

Example Graph (2)



In general, a cycle indicates there *may* be deadlock.

Cycles

- Suppose P_3 now requests R_2…
  - a request edge $P_3 \rightarrow R_2$ is added to the previous graph to show this

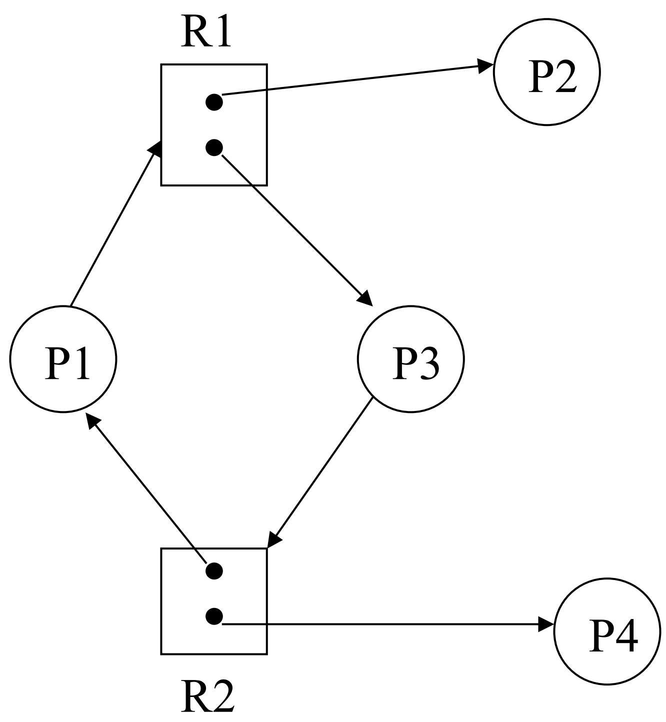
- There are now two cycles in the system:

$$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$$
$$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$$

- From this we can see that P_1, P_2, and P_3 are deadlocked

- Now consider the following the resource allocation graph…

Another Example



Deadlock?

Dealing with Deadlock

- Prevention
  – Devise a system in which deadlock cannot possibly occur
- Avoidance
  – Make decisions dynamically as to which resource requests can be granted
- Detection and recovery
  – Allow deadlock to occur, then cure it
- Ignore the problem
  – Common approach (e.g. UNIX, JVM)

Exercise

- Why might ignoring the problem of deadlock be a useful approach?

Deadlock Prevention

- Techniques
  - Force processes to claim all resources in one operation
    - Problem of under-utilisation
  - Require processes to claim resources in pre-defined order
    - e.g. tape drive before printer always
  - Grant request only if all allocated resources released first
    - e.g. transferring file from tape to disk, then disk to printer

Deadlock Avoidance

- Requires information about which resources a process plans to use

- When a request made, system analyses allocation graph to see if it may lead to deadlock
  - If so, process forced to wait
    - Problems of reduced throughput and process starvation

Deadlock Avoidance: Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves it in a safe state

- System is in such a state if for the sequence of processes <$P_1, P_2, \ldots, P_n$>, for each P_i, the resources that P_i can still request can be satisfied by currently available resources plus the resources held by all the P_j, with $j < i$.

- Thus:
  - If P_i resource requirements are not immediately available, then P_i can wait until all P_j have finished
  - When P_j is finished, P_i can obtain its required resources, execute, return allocated resources, and terminate
  - When P_i terminates, P_{i+1} can obtain its required resources,
  - … and so on

Deadlock Avoidance: Safe State

- If the system is in a safe state there are no deadlocks
- If the system is in an unsafe state, there is the possibility of deadlock
  - an unsafe state may lead to it
- Deadlock avoidance: ensure that the system will never enter an unsafe state
  - Avoidance algorithms make use of this concept of a safe state by ensuring that the system always remains in it

Detection and Recovery

- Systems that do not have deadlock prevention or avoidance mechanisms and do not want to ignore the problem must provide the following to deal with deadlock:
  - An algorithm to analyse the state of the system to see if deadlock has occurred
  - A recovery scheme

- Method depends upon whether or not there are multiple instances of each resource type…

Detection and Recovery

- If there are multiple instances of a resource type detection algorithms can be used that track:
  - the number of available resources of each type
  - the number of resources of each type allocated to each process
  - the current requests of each process
- If all resources have only a single instance, can make use of a wait-for graph
  - Variant of a resource-allocation graph
  - Obtained from resource allocation graph by removing nodes of type resource and collapsing the appropriate edges