Deadlock

Definitions

A process is *deadlocked* if it is waiting for an event that will never occur.

Typically, more than one process will be involved in a deadlock

A process is *indefinitely postponed* if it is delayed repeatedly over a long period of time while the attention of the system is given to other processes,

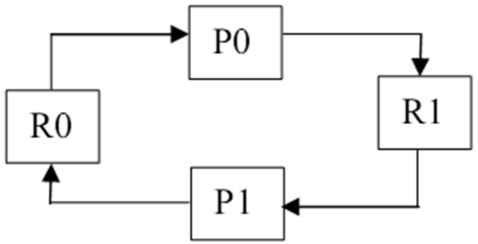
i.e. the process is ready to proceed but never gets the CPU

The Deadlock Problem

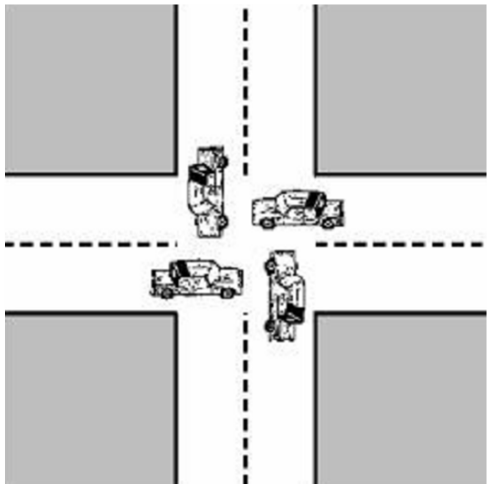
A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.

Example

System has 2 CD drives. P1 and P2 each hold one CD drive and each needs the other one.



a specific condition when two or more processes are each waiting for the other to release a resource, or more than two processes are waiting for resources in a circular chain



Resources

Resource

commodity required by a process to execute

Resources can be of several types

Serially Reusable Resources

CPU cycles, memory space, I/O devices, files

acquire -> use -> release

Consumable Resources

Produced by a process, needed by a process - e.g. Messages, buffers of information, interrupts

create ->acquire ->use

Resource ceases to exist after it has been used

Conditions for Deadlock

The following 4 conditions are necessary and sufficient for deadlock

Mutual Exclusion:

Only once process at a time can use the resource.

Hold and Wait:

Processes hold resources already allocated to them while waiting for other resources.

No preemption:

Resources are released by processes holding them only after that process has completed its task.

Circular wait:

A circular chain of processes exists in which each process waits for one or more resources held by the next process in the chain.

System Model

Resource types

$R_1, R_2, \ldots R_m$

Each resource type R_i has W_i instances

Assume serially reusable resources

request -> use -> release

Resource Allocation Graph

A set of vertices V and a set of edges E

V is partitioned into 2 types

P = {P1, P2,...,Pn} - the set of processes in the system

R = {R1, R2,...,Rn} - the set of resource types in the system

Two kinds of edges

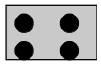
Request edge - Directed edge Pi ---> Rj

Assignment edge - Directed edge Rj ----> Pi

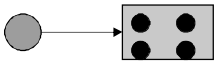
Resource Allocation Graph

Process

Resource type with 4 instances

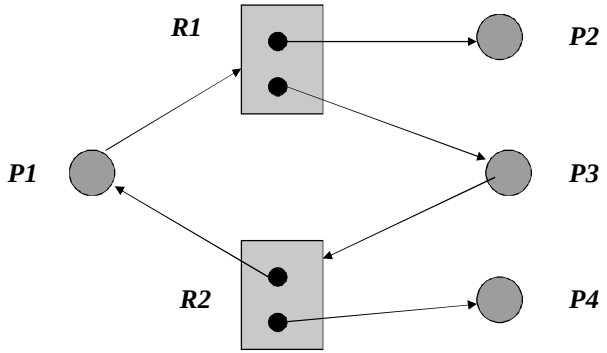


Pi requests instance of Rj

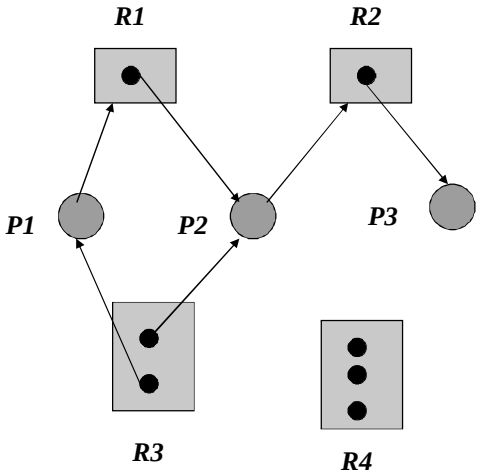


Pi is holding an instance of Rj

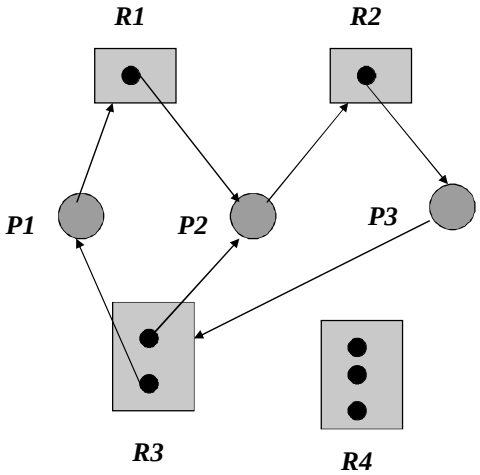
Graph with cycles



Graph with no cycles



Graph with cycles and deadlock



If graph contains no cycles

NO DEADLOCK

If graph contains a cycle

if only one instance per resource type, then deadlock

if several instances per resource type, possibility of deadlock.

Ignore The Problem

- Ostrich Algorithm
  - a strategy of ignoring potential problems on the basis that they may be exceedingly rare - "to stick your head in the sand and pretend that there is no problem". This assumes that it is more cost-effective to allow the problem to occur than to attempt its prevention.

Methods for handling deadlocks

1. Ignore the problem and pretend that deadlocks never occur in the system;

   Used by many operating systems, e.g. UNIX
2. Detection and Recovery
3. Avoidance
4. Prevention

Deadlock Detection

- Identify the processes and resources that have been deadlocked.
- Track the request and release of the resources.

Recovery from Deadlock: Process Termination

Abort all deadlocked processes.

Abort one process at a time until the deadlock cycle is eliminated.

In which order should we choose to abort?

- Priority of the process
- How long the process has computed, and how much longer to completion.
- Resources the process has used.
- Resources process needs to complete.
- How many processes will need to be terminated.
- Is process interactive or batch?

Deadlock Avoidance

Only allow access to the resource request that doesn't led to deadlock

Requires that the system has some additional information available.

- each process declare the maximum number of resources of each type that it may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
- Resource allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

Recovery from Deadlock: Resource Preemption

Selecting a victim - minimize cost.

Rollback

- return to some safe state, restart process from that state.

Starvation

- same process may always be picked as victim; include number of rollback in cost factor.

Deadlock Avoidance (Bankir's Algorithm)

- Set of resources, set of customers, banker
- Rules
  - Each customer tells banker maximum number of resources it needs.
  - Customer borrows resources from banker.
  - Customer returns resources to banker.
  - Customer eventually pays back loan.
- Banker only lends resources if the system will be in a *safe state* after the loan.

Safe state

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

System is in safe state if there exists a safe sequence of all processes and can satisfy all the request

Example

- Max system resources : 10
- There are 3 processes : A, B, C
- A needs max resources : 10, now is holding : 2
- B needs max resources : 3 , now is holding : 1
- C needs max resources : 7 , now is holding : 3
- Available resources : 4

Unsafe State

- System is in unsafe state if there no exists a safe sequence of all processes and can't satisfy all the request

PROCESS	ALLOCATED RESOURCES	MAX RESOURCES
A	2	10
B	1	3
C	3	7
FREE : 4		

SAFE STATE = ?
UNSAFE STATE = ?

If a system is in a safe state $\Rightarrow$ no deadlocks.

If a system is in unsafe state $\Rightarrow$ possibility of deadlock.

Avoidance $\Rightarrow$ ensure that a system will never reach an unsafe state.

Deadlock Prevention

If any one of the conditions for deadlock (with reusable resources) is denied, deadlock is impossible.

Restrain ways in which requests can be made

- Mutual Exclusion
  - non-issue for sharable resources
  - cannot deny this for non-sharable resources (important)
- Hold and Wait - guarantee that when a process requests a resource, it does not hold other resources.
  - Force each process to acquire all the required resources at once. Process cannot proceed until all resources have been acquired.
  - Low resource utilization, starvation possible

Deadlock Prevention (cont.)

No Preemption

- If a process that is holding some resources requests another resource that cannot be immediately allocated to it, the process releases the resources currently being held.
- Preempted resources are added to the list of resources for which the process is waiting.
- Process will be restarted only when it can regain its old resources as well as the new ones that it is requesting.

Circular Wait

- Impose a total ordering of all resource types.
- Require that processes request resources in increasing order of enumeration; if a resource of type N is held, process can only request resources of types > N.