1 DEADLOCK

A process is in a state of deadlock if it is waiting for an event which will not occur

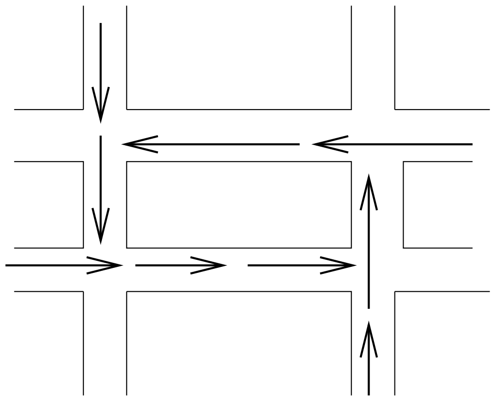

Figure 1: Traffic Jam

In OS context :

Shared resources

4 tape drives in system, and 2 processes. Each process holds 2 tape drives BUT needs 3 to progress

Spooling

Multiple processes spooling print files to disk

Printer waits for files to be closed before printing

Run out of disk space

File cannot be completed (closed) until something is printed to free some space - nothing can be printed until a file is completed !!

Start printing one file - or start killing processes to free space

2 Conditions for deadlock to occur

- **Mutual Exclusion** : At least one unsharable resource - processes claim exclusive control of resources they need
- **Hold and Wait** : Process *holds* one resource while waiting for another
- **No Preemption** : Resources only released voluntarily - no interruption possible (i.e. cannot be forcefully withdrawn by another process)
- **Circular Wait** : Circular chain of processes - each waiting for a resource held by another

Deadlock is a *SERIOUS* problem that can occur in concurrent systems, how do you - as OS designer and developer - *control deadlock* ?

Four strategies - we shall look at each in term :

- *Prevention*
- *Avoidance*
- *Detection*
- *Recovery*

3 Deadlock Prevention

Simple really - *DENY* one of the four necessary conditions!

1. Make resources sharable (e.g spooling) $\equiv$ *no mutual exclusion*

2. Processes MUST request ALL resources at the same time. Either all at start or release all before requesting more $\equiv$ *no hold and wait for* :
   Poor resource utilisation and possible starvation

3. If process requests a resource which is unavailable - it must release all resources it currently holds - and try again later $\equiv$ *allow preemption* - loss of work

4. Impose an ordering on resource types. Process requests resources in a pre-defined order $\equiv$ *no circular wait* = this can be over restrictive

In general therefore, with prevention we get *low device utilisation and reduced system throughput*

4 Deadlock Avoidance

Allow necessary conditions to occur, but use algorithms to predict deadlock and refuse resource requests which could lead to deadlock

4.1 Dijkstra's Bankers Algorithm

(encountered before in the process scheduling model)

- **Process need** = Maximum quantity of each resource that process requires during its lifetime.
- **Resource Manager (Banker)** = needs prior knowledge of all process claims
- Process requests *ONE* resource at a time
- Current process request is allowed if :
  the request plus current usage of the processes is less than the need
  after allocating the request there exists a sequence in which all processes can run to completion even if they demand their full claim - SAFE STATE
- Users must guarantee to release resources in finite time
- Resource manager guarantees to allocate resources in finite time Process

Example : 12 tape drives, 3 users

User	Current Loan	Max Needed
1	1	4
2	4	6
3	5	8

Available drives = 2
Allocate 2 drives to user 2
user 2 completes and releases 6 drives
allocate 3 drives each to users 1 and 3
ALL users run to completion = SAFE STATE

User	Current Loan	Max Needed
1	8	10
2	2	5
3	1	3

Available = 1
If grant last drive to *any user* may get deadlock = UNSAFE STATE
Existence of an UNSAFE state DOES NOT imply existence or eventual existence of deadlock, e.g. user 1 may release current loan before user 1 and 2 request extra.

4.2 Bankers Algorithm

- Processes request 1 resource at a time

- Request granted ONLY if it results in a safe state
- If request results in an unsafe state, it is denied and user holds resources and waits until request is eventually satisfied
- In finite time, all requests will be satisfied
- Can be extended to cater for multiple resource types

+ : No deadlock and not as restrictive as deadlock prevention
- : Fixed number of resources and users
Guarantees finite time - NOT - reasonable response time
Needs advanced knowledge of maximum needs
Not suitable for multi-access systems
Unnecessary delays in avoiding unsafe states which may not lead to deadlock

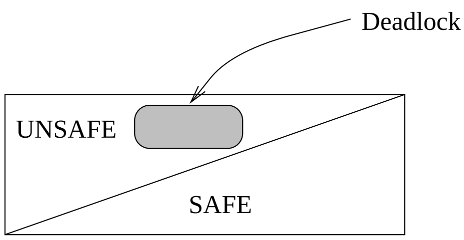

Figure 2: Bankers algorithm (safe/unsafe states)

Therefore, from Bankers algorithm :
Allow a thread (process) to proceed if :

(total available resources) - (number allocated)
$\geq$ (maximum remaining that might be needed by the thread (process))

5 The Dining Trumpet players

(aka *Dining Philosophers* - but I prefer trumpet players)
Designed by Dijkstra : used as a benchmark to test synchronisation primitives.

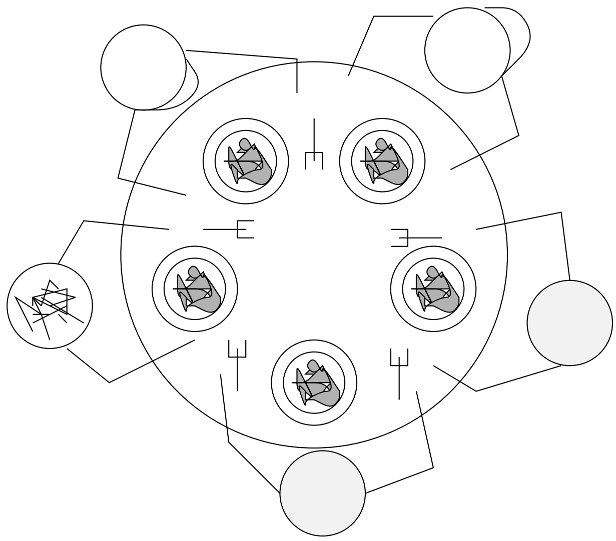

Figure 3: Dining Trumpet players)

Each needs two forks to eat - with alternate periods of eating and chatting.

- **Aim** : To pick up two forks (eat) and then put down the forks (talk) - and do so without getting *stuck*.
- Acquire left fork, then right fork - if everyone wants to eat simultaneously - we have deadlock!
- Some kind of time ordering - perhaps - is required

6 Deadlock Detection

- Permit first three necessary conditions for deadlock
- For greater efficiency, tolerate occasional deadlock
- Require early detection to avoid 'doing nothing'
- Take recovery action

Algorithm for this :

1. Maintain information on current allocation of resources to processes + outstanding resource requests
2. Check for circular wait *via resource allocation graphs, petri nets etc*

Invoking the algorithm :

- Each time resource is requested - *may lead to very high overheads*
- Periodically
- When resource utilisation drops below a limit

7 Deadlock Recovery

- **Abort all deadlock processes** and release resource - *too drastic* - will lead to loss of work
- **Abort one process at a time** - releasing resources until no deadlock
  How do we determine which process to abort first ? - *priority ordering*, *process which has done least work*
- **Selectively restart** processes from a previous *checkpoint* i.e. before it claimed any resources
  difficult to achieve - sometimes impossible
- **Successively withdraw resources** from a process and give to another process until deadlock is broken. How to choose which processes and which resources ?

1. Complex decisions due to the large number of processes present within a system
2. Difficult to automate
3. Use Operator to resolve conflicts - BUT this requires the operator to have skill and understanding of what processes are actually doing

8 Deadlock Summary

- Two main strategies : *Prevention/Avoidance* or *Allow to occur and then recover*
- Different strategies for different resources
- Internal Resources : Internal data structures (like PCBs (Kernel)), *Prevention by resource ordering*
- Main Memory : Memory used by a job, *Prevention by preemption as requesting job can be swapped out and release memory it was holding*
- Job Resources : Dedicated devices such as tape drives, disks. *Avoidance since information needed about resource requirements can be known in advance*
- Swap Space : Disk Space for paging etc. *Avoidance by pre-allocation of maximum requirements, which are usually known in advance*
- File Space : *Operator recovery by selectively deleting files - temporary file, archived ones etc*