UNIVERSITY *of* STIRLING

Concurrent and Distributed Systems

Deadlocks II

Conditions for Deadlock

Reminder:

- Mutual exclusion condition
  - each resource assigned to 1 process or is available
- Hold and wait condition
  - process holding resources can request additional resources
- No preemption condition
  - previously granted resources cannot forcibly taken away
- Circular wait condition
  - must be a circular chain of 2 or more processes
  - each is waiting for resource held by next member of the chain

- All 4 conditions need to be meet for deadlocks to occur
- Conditions can be modelled using graphs

Strategies for dealing with deadlock

- Ways of dealing with deadlocks

1. Ostrich algorithm (never)
2. Prevention, by negating one of the 4 conditions for deadlocks to occur (before)
3. Dynamic avoidance, by careful resource allocation (during)
4. Detection and recovery (after)

3. Deadlock Dynamic Avoidance

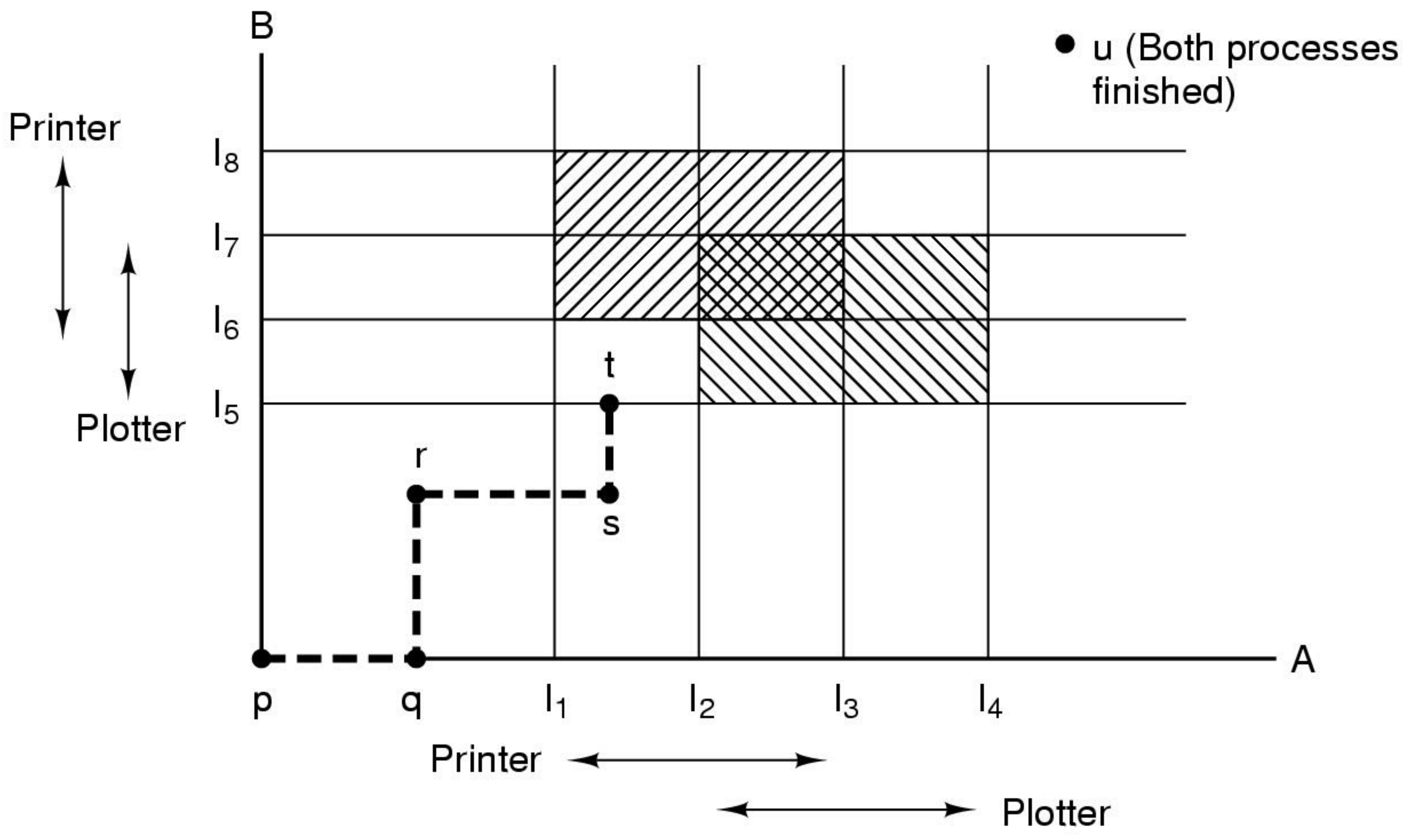
Based on dynamically managing resource allocation

- Resources are requested one at a time
- To avoid deadlocks, OS must decide whether it is safe or not to allocate a resource and only allocate if it is safe
- Is there an algorithm that can avoid deadlock by making the right choice?
- Yes – but certain information needs to be made available (overhead)
- The algorithm is based on safe and un-safe states.

3. Resource Trajectories

- Algorithm is based on safe states
- Look at the problem in an easy to understand and graphic way first
- Graphical approach does not translate directly into an algorithm but provides a good sense what is required

3. Resource Trajectories - Example



3. Safe and Unsafe States

- Based on the same resource matrixes used for the detection algorithm (more later)
- State is safe if
  - it is not deadlocked
  - There exists a possible sequence of resource allocations that allows each process to acquire all the needed resources, according to their maximum limit, and hence terminate and release the resources.

3. Safe and Unsafe States

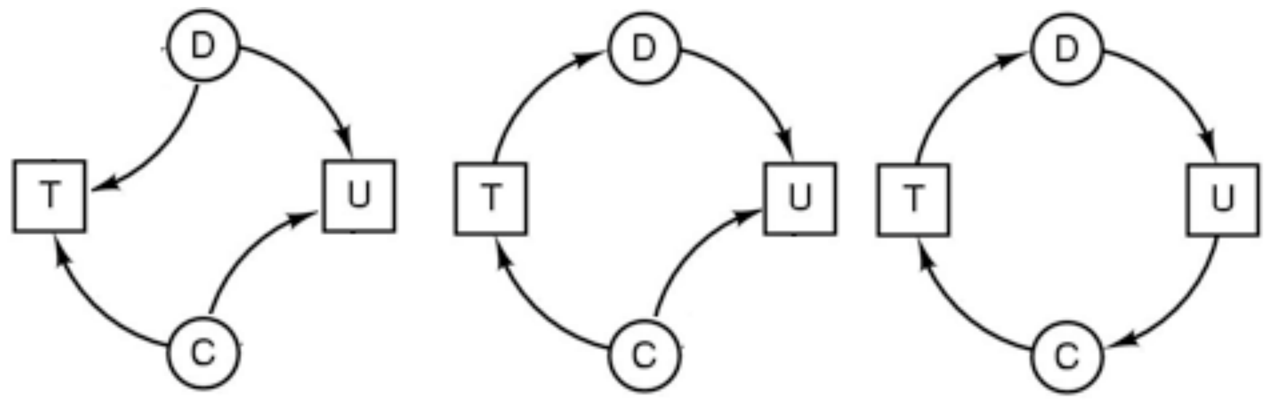
- An unsafe state is not a deadlocked state!
  - It is a state that will eventually lead to a deadlock, if no resources are freed!

- Safe state: it is guaranteed that all processes will terminate

- Unsafe state: it cannot be guaranteed that all processes will terminate

3. Deadlock Dynamic Avoidance

Example (sketch):

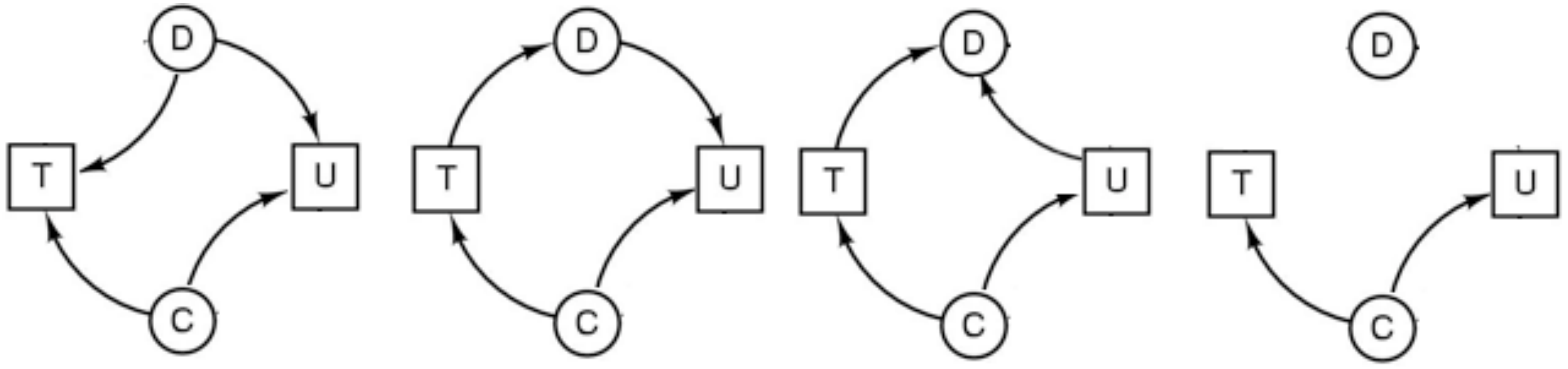


1 **2** **3**

1. Processes D and C need resources U and T
2. T is granted to D, the state is safe (as we will see in a minute)
3. U cannot be granted to C, as the resulting state would not be safe, actually would be deadlocked, as shown ...

3. Deadlock Dynamic Avoidance

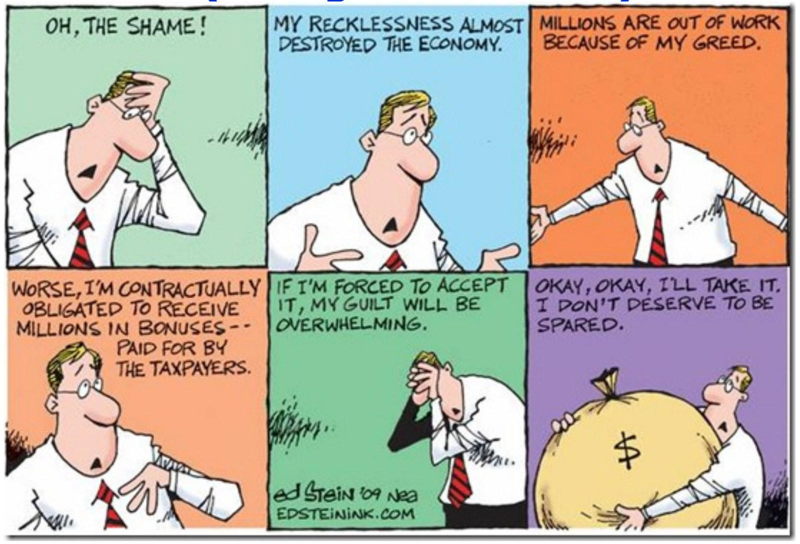
Example (sketch):



1 **2** **3** **4**

1. Processes D and C need resources U and T
2. T is granted to D, the state is safe (as we will see in a minute)
3. U cannot be granted to C, as the resulting state would not be safe, actually it is deadlocked ... and indeed U is also granted to D
4. D has all the needed resources, will eventually terminate and make them available to C.

3. Banker's Algorithm (Dijkstra)


OH, THE SHAME!
MY RECKLESSNESS ALMOST DESTROYED THE ECONOMY.
MILLIONS ARE OUT OF WORK BECAUSE OF MY GREED.
WORSE, I'M CONTRACTUALLY OBLIGATED TO RECEIVE MILLIONS IN BONUSES-- PAID FOR BY THE TAXPAYERS.
IF I'M FORCED TO ACCEPT IT, MY GUILT WILL BE OVERWHELMING.
OKAY, OKAY, I'LL TAKE IT. I DON'T DESERVE TO BE SPARED.
$
ed Stein '09 NEA
EDSTEININK.COM

- A deadlock avoidance algorithm (see detection algorithms – later)
- Based on a small-town banker and his dealings with customers
- Algorithm denies or delays any resource request that leads the system to an unsafe state …
- Resources may be allocated to a process only if requested resources are less than or equal to available resource; otherwise, the process waits until resources are available.
- … when a process has obtained all the resources it needs, it can terminate and must eventually free all resources

3. Banker's Algorithm Multiple Res.

Process	Tape drives	Plotters	Scanners	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Scanners	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)
P = (5322)
A = (1020)

E = existing resources

A= available resources

P= assigned resources

D could be granted one of the two available scanner, terminate and release all possessed resources. There are not sufficient resources for A at the moment – although it could still acquire a Tape driver.

If all processes are in the same state as A deadlock will occur.

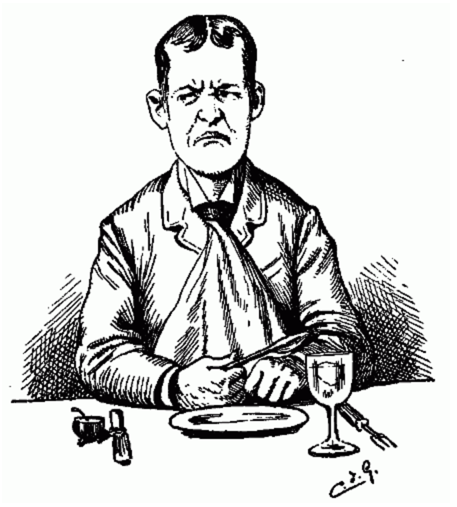
3. Drawbacks of the Banker's Algo.

- Theory is wonderful, however …

- Processes rarely know their maximum number of resources in advance
- Number of processes in the system is not fixed, but dynamically changes over time
- New users log on to a system
- Previously available resources may vanish suddenly (break)
- There are no guarantees on when resources will be released.

- As a consequence, few systems actually use the banker's algorithm

Starvation

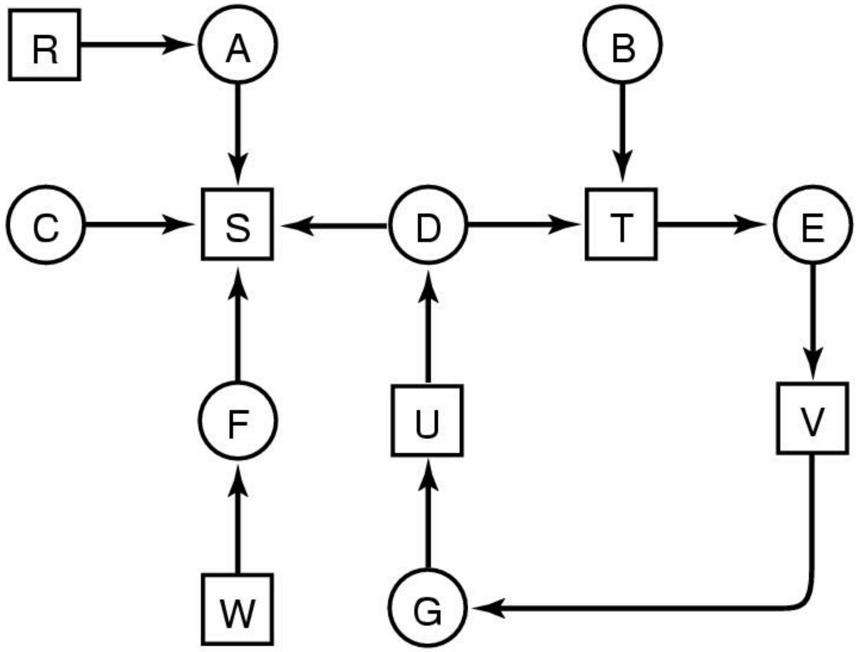


- Algorithm to allocate a resource
  - May be to give to shortest job first
- Works great for multiple short jobs in a system
- May cause long job to be postponed indefinitely
  - Even though not blocked
- Solution
  - First–come, first-serve policy

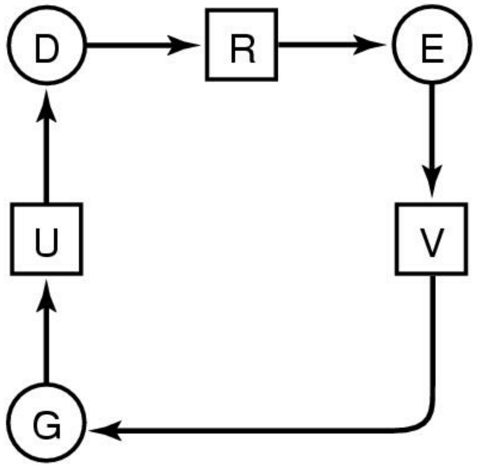
4. Detection and Recovery

- Simplest case: a system has only one resource of each type (1 printer, 1 plotter)
- Construct a resource graph as discussed before
- If the graph contains any cycles → deadlock
  - Any process part in a cycle is deadlocked

Example



(a)



(b)

- Simple to detect deadlocks in a graph (polynomial)
- However, formal algorithm required for actual implementation → much more complex

4. Detection with multiple Resources

- Matrix based algorithm, n processes, m resources
- A resource is either allocated or is available

Resources in existence
$(E_1, E_2, E_3, \ldots, E_m)$

Resources available
$(A_1, A_2, A_3, \ldots, A_m)$

Current allocation matrix

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$$

Row n is current allocation to process n

Request matrix

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Row 2 is what process 2 needs

4. Detection with multiple Resources

$$E = \begin{array}{cccc} A & B & C & D \\ (4 & 2 & 3 & 1) \end{array} \qquad A = \begin{array}{cccc} A & B & C & D \\ (2 & 1 & 0 & 0) \end{array}$$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

- What if process 3 also requires a resource D?

4. Detection with multiple Resources

Deadlock detection algorithm:

The algorithm finds a process where the request for resources can be met. It then assumes it runs to completion, and so releases all its resources making them available.

1. Look for an unmarked process, P_i - for which the i-th row of **R**equest matrix is less than or equal to **A**vailable.
2. If such a process is found, add the i-th row of **C**urrent allocation matrix to **A**vailable (Pi can acquire needed resources, terminate and make all them available), and mark the process, and then go back to step 1.
3. If no such process exists, the algorithm terminates.
4. One or more unmarked rows show deadlock exists.

4. When to check for deadlocks?

- Every time a resource request has been issued in the system → detects deadlocks as early as possible *but* very costly
- Check every *i* minutes
- Check only when the CPU utilisation drops below a certain threshold (only a few processes are running)

4 Recovery through Pre-emption

- Take a resource away from a process and assign it to another process (manual intervention may be needed)
- Issues:
  - Selection of a victim
  - Order pre-emption to minimise cost
  - Cost includes: number of resources held, time the process has already run
  - Rollback
    - What to do with the victim? Kill it or ‘rollback’ to a saved state
  - Starvation
    - Resources may be pre-empted by always the same process
    - Processes may be picked only a finite number of times

4 Recovery through Rollback

- checkpoint a process periodically, save state
- State should contain memory image and status of resources
- use this saved state
- restart the process if it is found deadlocked with the state of the last check point

- Work since the used checkpoint is lost
- Process has to wait to re-acquire the resource

4. Recovery through Killing Processes

- Crudest but simplest way to break a deadlock
- State of some resources may be incorrect
  - E.g. updating a file
- Kill all processes in the deadlock
  - Certainly resolves the deadlock
  - Very expensive, all processes need to re-run
- Kill one process at a time
  - choose process that can be re-run from the beginning
  - Incurs overhead, after a process is aborted the deadlock detection algorithm needs to be run again.
- the other processes get the resources of the killed process

4. Select processes for termination

- What is the priority of the process?
- How long has the process computed, and how much longer the process will compute to finish its task?
- How many and what resources the processes holds (simple to pre-empt?)?
- How many more resources a process needs to complete computation?
- How many processes will need to be terminated?
- Whether the process is interactive or batch?

Deadlock handling: summary

1. **Ostrich algorithm**
   - do nothing.

2. **Prevention**:
   1. **Mutual Exclusion**
   2. **Hold and Wait**
   3. **No Preemption**
   4. **Circular Wait**

3. **Dynamic avoidance**:
   - safe/unsafe states
   - Banker algorithm.

4. **Detection and recovery**:
   1. **Detection**
   2. **Preemption**
   3. **Rollback**
   4. **Killing processes**