UNIVERSITY *of* STIRLING

Concurrent and Distributed Systems

Deadlocks I

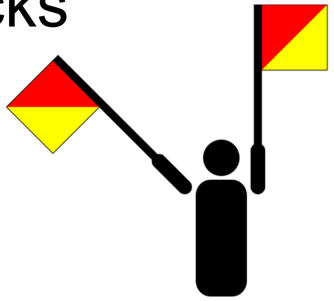
Contents

Kansas train crossing law 1900s

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

Deadlocks

- Previously we discovered examples of deadlocks
  - Semaphore
  - Java Synchronisation

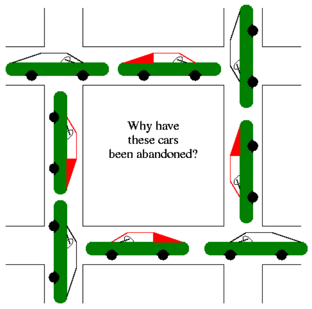


- Deadlocks are associated with acquiring a resource
- Resources
  - Are often non-sharable:

    can be used by one process at a time
  - Can be both hardware and software
  - Examples: network interfaces, printers, system tables, database entries, memory locations

Deadlocks

- Processes need to acquire resources in reasonable order
  - Processes sometimes require more than one resource
  - Order in which the resources are allocated is important
- A process holds resource A and requests resource B
  - Another process holds resource B & requests resource A
  - Both processes are blocked and remain so forever
- Deadlocks can occur across machines
  - Shared devices between a number of machines
  - Printers, scanners, CD recorders
  - Resources can be reserved remotely



Resources

- Deadlocks can occur if processes have been granted exclusive access to resources
  - A system contains many resources …
  - … and number of instances of the same type (printers)
- Deadlocks can involve any number of processes and resources

Non-preemptable Resources

- Preemption may cause the process to fail
  - Example: CD burner, if started to burn and taken away and given to another process → garbage on the CD
- Generally deadlocks involve non-preemptable resources
- Potential deadlocks with preemptable resources can be avoided by reallocating the resources

Resource usage

- Sequence of events to use a resource
  - Request a resource
  - Use the resource
  - Release the resource
- If a resource is not available when requested the process is forced to wait
- Use of some synchronisation mechanisms
  - Semaphore, Monitors, etc
- Processes may need more than one resource
  $\rightarrow$ the order for acquiring resources may be relevant

Example: acquiring resources

```
Semaphore res1;
Semaphore res2;

void processA(void) {
   p(res1);  p(res2);
   use_the_resources();
   v(res2); v(res1);
}
void processB(void) {
   p(res1);  p(res2);
   use_the_resources();
   v(res2);  v(res1);
}
```

```
Semaphore res1;
Semaphore res2;

void processA(void) {
   p(res1);  p(res2);
   use_the_resources();
   v(res2); v(res1);
}
void processB(void) {
   p(res2);  p(res1);
   use_the_resources();
   v(res1);  v(res2);
}
```

Definition of Deadlocks

- Formal definition : A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause

- Usually the event is the release of a currently held resource
- It Is a permanent condition, none of the processes can …
  - run
  - release resources
  - be awakened
- Number of resources and processes is unimportant
- Kind of resources is unimportant

Conditions for Deadlock

- 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 met for deadlocks to occur
- Conditions can be modelled using graphs

Definition of Deadlocks

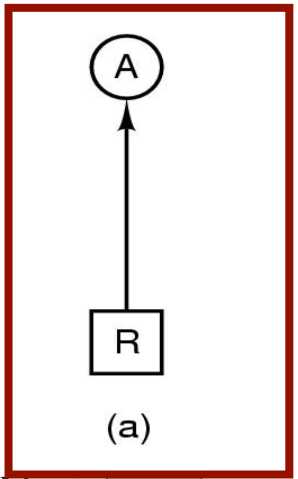
- Formal definition : A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause

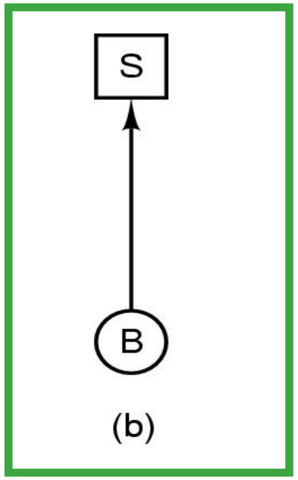
It implies, informally speaking, the four conditions:

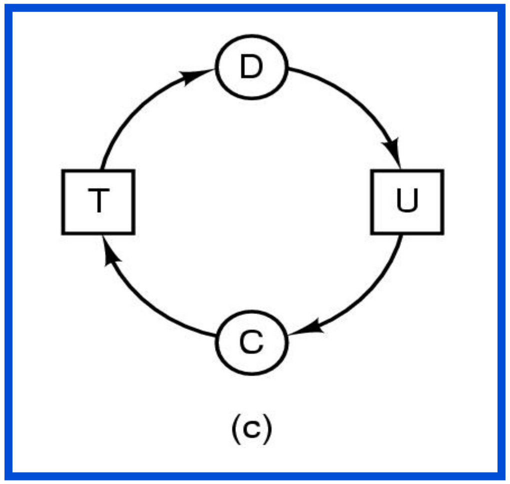
- only processes in the set can release resources, no preemption
- no sharing, the possession of one resource is “blocking”
- no release of resources (all processes are waiting, can not do other actions)
- the definition implies that at least two processes are in a circular wait.

Deadlock Modelling

- Graphs have two kind of nodes: processes (circles) and resources (squares)
- An arc from a resource to a process means the resource as previously been assigned (a)
- And arc from a process to a resource means the process is requesting this resource (currently blocked) (b)
- A cycle means deadlock (c)







Strategies for dealing with deadlock

The problem of deadlock can be dealt with by several different techniques:

1. **Ostrich algorithm**: ignore the problem.
   Maybe if you ignore it, it will ignore you.

2. **Prevention**: by negating one of the four conditions necessary for deadlocks to occur "BEFORE"

3. **Dynamic avoidance**: by careful resource allocation "MEANWHILE"

4. **Detection and recovery**: allow deadlocks to occur, detect them, and take some action "AFTER"

1. The Ostrich Algorithm

- Pretend there is no problem,

  i.e. do not put in place any strategy, control, … it will happen rarely, in case “restart”.

- Mathematicians vs engineers
- Reasonable *if* ….
  - deadlocks occur very rarely
  - cost of prevention is high
  - no huge consequences if deadlock occurs
- It is a trade-off between
  - convenience
  - Correctness
- UNIX and Windows takes this approach

2. Deadlock Prevention

- Deadlock avoidance is very hard!
- Prevention considers the four conditions for deadlock and attacks them:

2.1 Attacking the Mutual Exclusion Condition

2.2 Attacking the Hold and Wait Condition

2.3 Attacking the No preemption Condition

2.4 Attacking the Circular Wait Condition

2.1 Attacking Mutual Exclusion

- If no resource would ever be exclusively assigned to any process, no deadlock would be possible
  - However, two processes write to the same printer?
- Some devices (such as printer) can be spooled
  - only the printer daemon uses printer resource
  - thus deadlock for printer eliminated

- Not all devices can be spooled (process table)
- Does not provide 100% deadlock freedom

2.2 Attacking Hold and Wait

- Require processes to request all resources before starting execution
  - All resources are assigned at the beginning
  - A process never has to wait for what it needs
  - Processes always run to completion and release resources

- Problems
  - Processes may not know required resources at start of run
  - also ties up resources other processes could be using
  - Not optimal use of resources!
- Variation:
  - Processes must give up all resources before acquiring a new one
  - Then request all immediately needed resources again

2.3 Attacking No Preemption

- Not very promising!
- Some resources cannot be preempted
- Imaging a process accessing a CD writer
  - halfway through its job
  - now forcibly take away CD writer
  - !!??

2.4 Attacking Circular Wait

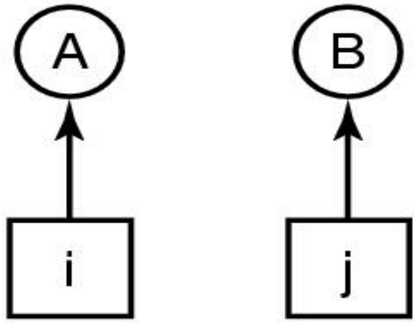
- Several ways of attacking possible

- A process is only entitled to a single resource
  - If a second resource is needed, the first resource has to be freed
  - Imagine a process copying a big file from tape to printer: unacceptable

- Global numbering of all resources
  - Processes can request resources whenever they want
  - Processes may hold as many resources as needed
  - BUT requests must be made in numerical order: strong constraint!

Attacking Circular Wait

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD Rom drive



- Each process acquires resources in ascending order
- The process holding the resource with the highest number associated cannot be waiting for any other resource (before freeing the ones currently held),
- hence that process can not be part of a circular wait.
- More in general, there cannot be cycles in the resource graphs, hence no deadlock can occur.
- Processes will eventually finish, hence freeing their resources.

Summary of deadlock prevention

Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically

Summary

- More on Deadlocks
- How can deadlocks be modelled
- What can be done about deadlocks
- Ostrich Algorithm
- Deadlock Prevention
- Next Lecture on Deadlock Avoidance and Detection and Recovery + examples