Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE:	**Cambridge (01223) 515010**
INTERNATIONAL:	**+44 1223 515010**
FAX:	**+44 1223 359779**
E-MAIL:	**apm@ansa.co.uk**

Training

ANSAwise - CORBA Concurrency and Transactions

Chris Mayers

Abstract

Organizations wish to make efficient use of the hardware and software they have purchased and intend to purchase.

Concurrent programming techniques can result in more efficient use of resources, and more rapid responses to invocations. However, concurrent programs are difficult to write correctly.

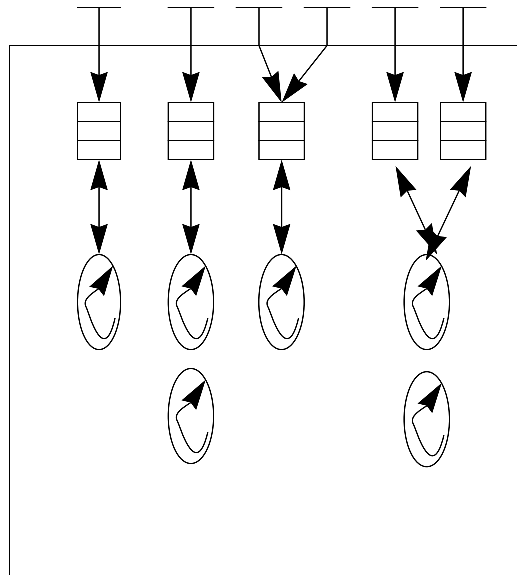
This module of the ANSAwise training programme explains the idea of lightweight threads. It then discusses the CORBA Concurrency and Transactions Object Services

APM.1540.02	**Approved**	2nd April 1996
	Briefing Note	

Distribution:
Supersedes:
Superseded by:

Copyright © 1996 Architecture Projects Management Limited

CORBA Concurrency and Transactions



 © Copyright 1996 Architecture Projects Management Limited

In this session

- Review the basic principles of concurrency
  - and the standards that are emerging
- Explain the CORBA Concurrency Object Service
  - and show how it relates to the CORBA Transactions Object Service

© Copyright 1996 Architecture Projects Management Limited

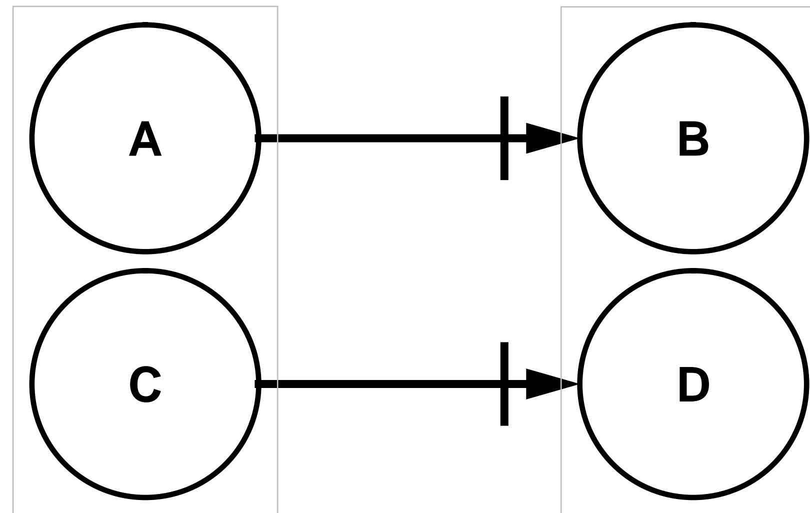
What is concurrency?

- Concurrency allows an application to overlap work
  - when one activity is blocked waiting for a response, other activities can execute
- Concurrency allows more efficient use of resources
  - but does not in itself guarantee a real-time response
  - nor does it control the use of resources

 © Copyright 1996 Architecture Projects Management Limited

Some concurrency comes automatically

- In a distributed system, some concurrency is inherent
  - when they are *not* communicating, client and server can execute ‘in parallel’ with each other...

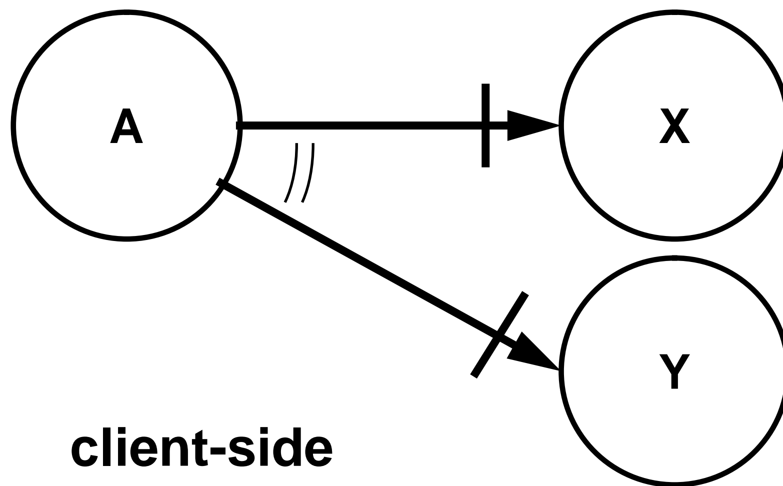


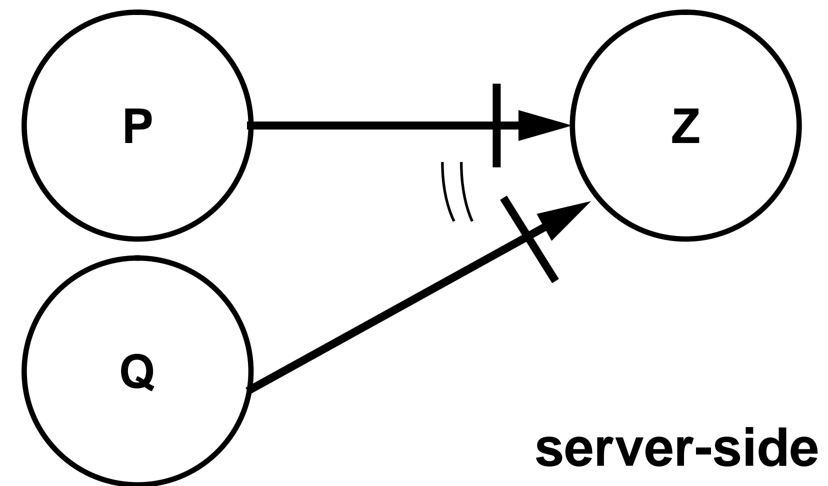
  - ... this is transparent

© Copyright 1996 Architecture Projects Management Limited

Client and server concurrency

- For efficiency, we must also consider concurrency
  - within a client and within a server





- Concurrency is primarily a design and implementation issue

© Copyright 1996 Architecture Projects Management Limited

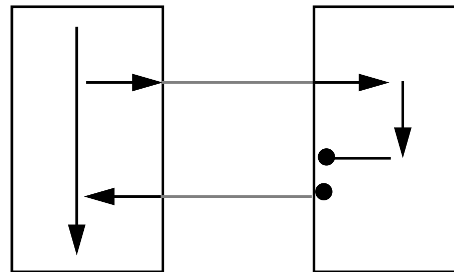
Before you start...

- Concurrency is usually not transparent
- Most programming languages (including C and C++) provide no support for concurrency
  - it is usually supported by an operating-system library instead
- CAUTION: concurrent programs are inherently difficult to test and debug
  - it is easy to make mistakes, and hard to detect them
  - only use concurrency if you need it

 © Copyright 1996 Architecture Projects Management Limited

Client concurrency

- Client concurrency involves submitting multiple requests...
  - ... one client invoking multiple servers at the same time
- Follow-up (asynchronous) RPC is one technique for client concurrency



- Use client concurrency when the application is ‘naturally parallel’
  - splitting the problem into several independent parts for replicated servers
  - ... numerical computation, indexing and searching, for example

 © Copyright 1996 Architecture Projects Management Limited

Server concurrency

- Server concurrency involves processing multiple requests
  - ... one server being invoked by multiple clients at the same time
- Concurrency is always an issue for servers
  - ... every object must have a concurrency policy
  - ... even if it is to handle only one request at a time (‘no concurrency’)

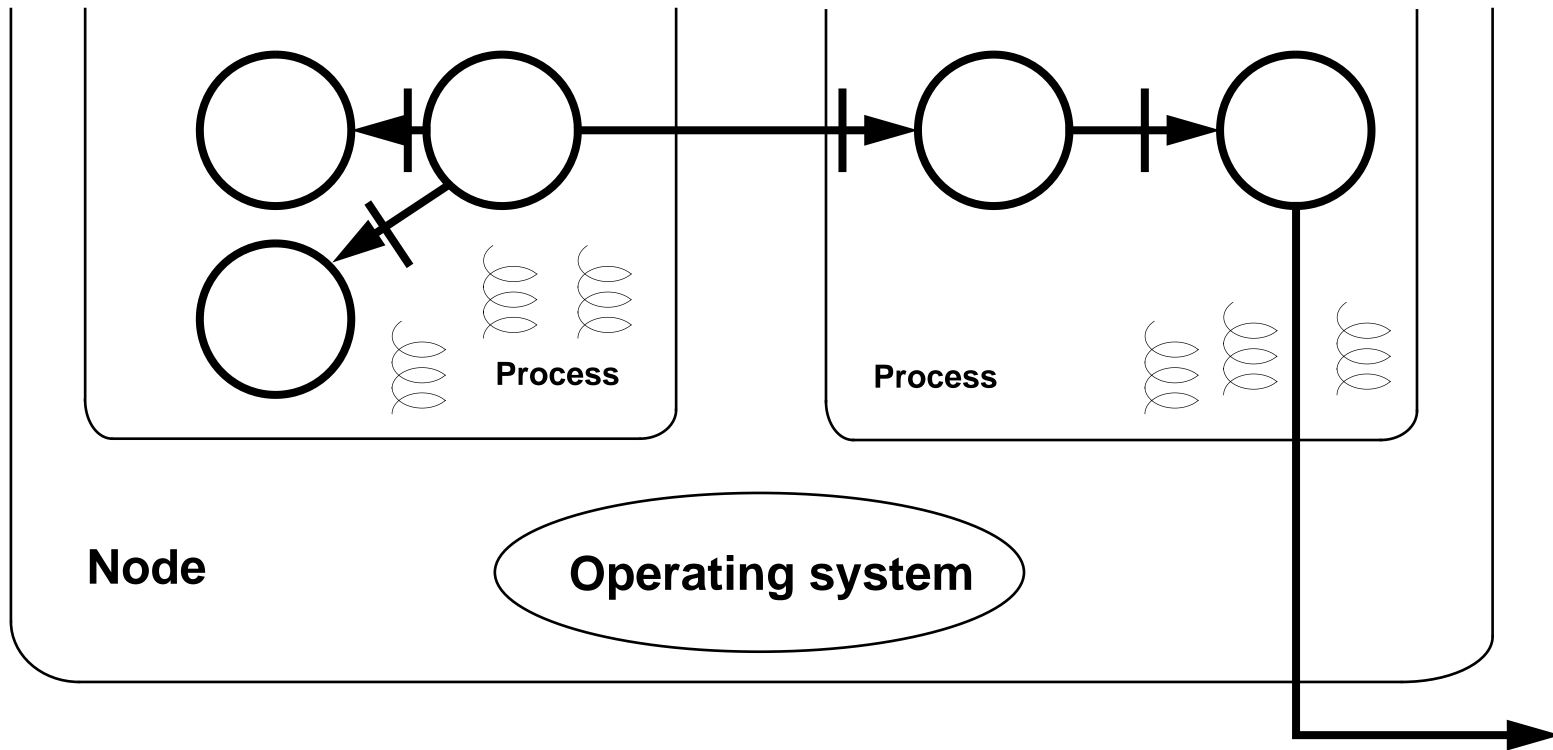
 © Copyright 1996 Architecture Projects Management Limited

Efficient concurrency

- Most operating systems support multiple processes
  - separately scheduled
  - not sharing memory
- These are called ‘heavyweight processes’
  - each process requires a lot of (scarce) operating system resources
- To support many concurrent activities efficiently, we need ‘lightweight processes’ (LWPs) within a heavyweight process
  - separately scheduled
  - sharing memory
  - requiring minimal resources per LWP
- LWPs can be implemented as *threads*

© Copyright 1996 Architecture Projects Management Limited

Typical system infrastructure



 © Copyright 1996 Architecture Projects Management Limited

Support for concurrency

- *Multithreading*: supporting multiple potential execution within a process

- *Multitasking*: scheduling actual processor time to threads

- *Multiprocessing*: supporting simultaneous execution on multiprocessor systems (tightly-coupled or clustered CPUs)

© Copyright 1996 Architecture Projects Management Limited

User-level and kernel-level thread implementations

- Threads can be supported either
  - within a system library that is linked with the application (user-level)
  - within the operating system’s kernel (kernel-level)
- What are the advantages and disadvantages of each approach?

Get ready to discuss this

 © Copyright 1996 Architecture Projects Management Limited

Your notes

© Copyright 1996 Architecture Projects Management Limited

Concurrency issues for servers

- **Isolation**
  - **avoiding interference between concurrent invocations by synchronization**
- **Organization**
  - **how the implementation’s use of concurrency is organized**

 © Copyright 1996 Architecture Projects Management Limited

Isolation and Shared Data

- Because two threads in the same process share memory, they can inadvertently corrupt each other's data
  - minimize the risk by sound software engineering practice
  - ... avoid global data
  - ... make it read-only where possible
  - ... exploiting relevant programming language features
- However, some data represents true shared state
  - it must be accessed safely...
  - ...access must be synchronized

© Copyright 1996 Architecture Projects Management Limited

Synchronization and deadlock

- Most systems use locks to avoid interference
- If two invocations each need access to shared data that the other invocation has locked, they will deadlock
- Most systems do not detect deadlocks
- Understanding and avoiding deadlocks is a complex topic

© Copyright 1996 Architecture Projects Management Limited

Synchronization and scheduling

- Scheduling may be either
  - *pre-emptive*: a thread may be suspended without knowing, and another thread scheduled
  - *non-pre-emptive*: a thread is only suspended when it explicitly yields to other threads
- Applications cannot generally assume either scheduling policy, and should allow for both
  - policies vary between systems

© Copyright 1996 Architecture Projects Management Limited

Allowing for scheduling

- To allow for pre-emptive scheduling:
  - threads must use synchronization mechanisms to ensure exclusive access to shared data
- To allow for non-pre-emptive scheduling:
  - a thread that executes for a long time (in a tight loop), should yield to allow other threads a chance to execute

 © Copyright 1996 Architecture Projects Management Limited

Thread Synchronization Mechanisms

- There is more than one mechanism that can provide a lock

- Operating systems support a large variety of synchronization mechanisms

  - spin locks, lockouts, mutexes, mutants, eventcounts/sequencers, events, event flags, flags, critical sections, semaphores, test-and-sets, zones,...

  - ... all non-portable, apparently similar, but subtly different

© Copyright 1996 Architecture Projects Management Limited

Thread standardization

- Standardization is emerging...
  - for thread control, scheduling, and synchronization mechanisms
- ... the most important is the POSIX Threads interface (pthreads) 1003.1c (formerly 1003.4a)
- DCE includes a slightly modified form of pthreads - DCE Threads
- The pthreads interface is an API; it is not object-oriented

© Copyright 1996 Architecture Projects Management Limited

Threads in CORBA?

- CORBA has separate Threads and Concurrency Object Services for threads and their synchronization mechanisms
  - unlike POSIX Threads, where they are combined
- CORBA does not yet specify an Object Service for threads
  - it is in the Object Services Roadmap, but there is no work in progress
- Thread support would have on impact on all implementations
  - of the ORB
  - of Object Services
- Some vendors supply multi-threaded ORBs as product options
  - but these may not use the POSIX Threads interface

© Copyright 1996 Architecture Projects Management Limited

Server Thread Creation

- Servers do not need to create threads explicitly
  - the distributed processing environment creates them automatically for each invocation
- Servers usually have some control over the number of concurrent threads created this way
  - by specifying a maximum number of threads...
  - ... a simple form of resource control
- Systems differ; this control may be
  - per capsule/per process
  - per object
  - per interface

© Copyright 1996 Architecture Projects Management Limited

Thread organization for explicit threads

- Applications can use explicit threads to gain more concurrency
- There are some conventional ways to organize these threads
  - boss/worker organization
  - work crew organization
  - pipelined organization
- These are design guidelines for server implementations
  - the organization is not visible to clients
  - the thread system is not aware of it

© Copyright 1996 Architecture Projects Management Limited

Boss/worker organization

- A boss thread assigns tasks to a pool of worker threads
  - either the workers call the boss for new work
  - ... or the boss polls the workers to see if they are free
  - alternatively, the boss and workers can interface via a work queue
- Analogy: the traditional typing pool

 © Copyright 1996 Architecture Projects Management Limited

Work crew organization

- The work can be divided into independent activities
  - each carried out by a separate thread executing in parallel
  - each usually does a different job
- A master thread creates these threads and waits for them to complete
  - the master thread can do work itself
- Analogy: painting a house

 © Copyright 1996 Architecture Projects Management Limited

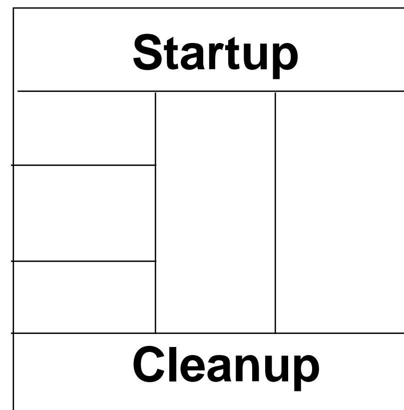
Pipeline organization

- The work can be divided into short independent activities
  - each dependent on the output of the previous one
- Analogy: assembly line

© Copyright 1996 Architecture Projects Management Limited

Compound organization

- These organizations can be combined...



- ... a work crew with an internal pipeline

© Copyright 1996 Architecture Projects Management Limited

When to use explicit threads

- Clients and servers can both use explicit threads...
  - ... but should only do so if the throughput is really needed
  - ... and the thread organization naturally matches the structure of the problem
- Synchronization is even more tricky than with implicit threads
  - parent threads must also keep track of child threads...
  - ... including handling the failure of child threads
  - ... and avoiding deadlock with them

© Copyright 1996 Architecture Projects Management Limited

CORBA Concurrency Control Object Service

- Enables coordination of access to shared resources
- The Concurrency Control Object Service does not define what a resource is
  - this is determined by the application
- Supports conventional locks
  - all the usual modes, including intention locks
- Supports transactional and non-transactional modes of operation
  - transactional mode: the CORBA Transaction mode drives the release of locks when transactions commit or abort
  - non-transactional mode: applications are responsible for releasing locks

 © Copyright 1996 Architecture Projects Management Limited

CORBA Concurrency Control Service

- The overall structure is

```
#include <CosTransactions.idl>
module CosConcurrencyControl {
     enum lock_mode {
        read,
        write,
        upgrade,
        intention_read,
        intention_write
     };
     ...
     interface LockCoordinator ...
     interface LockSet ...
     interface TransactionalLockSet ...
     interface LockSetFactory...
}
```


 © Copyright 1996 Architecture Projects Management Limited

Implications of the CORBA approach

- All the lock sets are CORBA objects
  - so can be accessed remotely, and used anywhere a CORBA object can be used...
  - ...even between ORBs
- Specialized implementations of the Concurrency Control service are likely
  - for efficiency
  - for tight integration with the CORBA Transactions service
  - for integration with legacy concurrency control mechanisms

© Copyright 1996 Architecture Projects Management Limited

Summary

- **Concurrency can give more efficient use of resources**
- **Portable concurrency with CORBA would require a Threads Object Service**
  - **vendor-specific implementations are becoming available**
- **For more information:**
  - **on the Concurrency and Transaction services, see *CORBAservices* by the Object Management Group (Wiley)**
  - **on designing and using concurrency, see *Transaction Processing: Concepts and Techniques*, by Jim Gray and Andreas Reuter (Morgan Kaufmann)...**
  - **... a general approach, not just for TP systems**

© Copyright 1996 Architecture Projects Management Limited