Concurrency Models in (RT) Middleware

Nanbor Wang
nanbor@cs.wustl.edu
http://www.cs.wustl.edu/∼nanbor/

October 13, 1999

Overview

- Basic concurrency strategies
- Various concurrency architectures
- TAO ORB Core

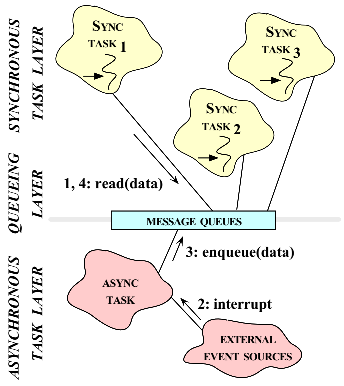
Cooperative/Asynchronous

- Multiple event handlers sharing a thread of control
- An event demultiplexing mechanism distributes events among handlers (e.g. `select ()`)
- Event handlers must carry their own states, hard to program
- Events should be handled “quickly”
- No overhead of context switching
- Reactor pattern

Concurrent/Synchronous

- A single thread of control handles a task synchronously
- Easy to program — procedural
- States are kept in threads’ stack
- OS is responsible for scheduling
- Creation and context switching are not free
- Synchronized access to shared resources could be expensive
- Performance improvement on multiprocessor platforms
- Active object pattern

The Half-Sync/Half-Asynch Pattern

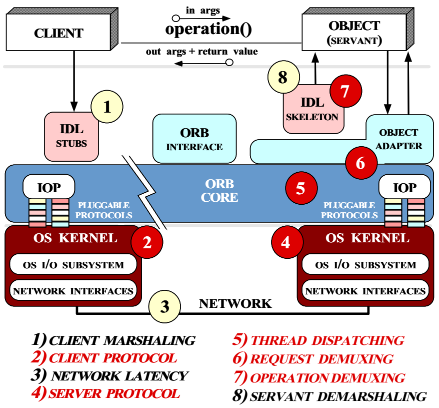


- Bridge the asynchronous event sources and synchronous processes
- Synchronous task layer performs higher level jobs
- Queueing layer provides synchronization and buffering
- Asynchronous task services external events

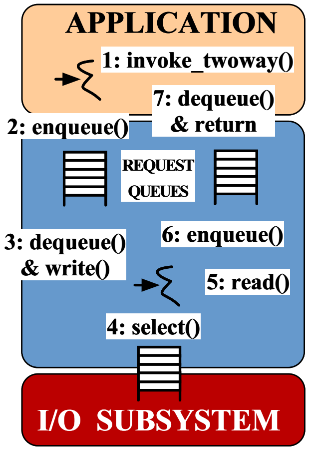
Half-Sync/Half-Async (Cont.)

- Penalty incured at cross boundary
  - Synchronization
  - Data copying
  - Context switching
- External events are serviced serially

Object Request Broker

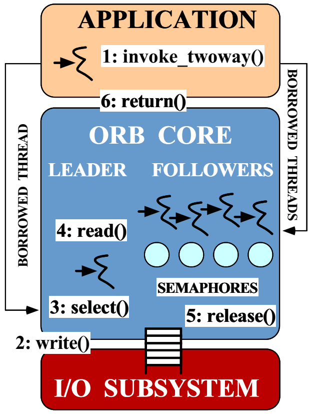


Active Connection – Client Side



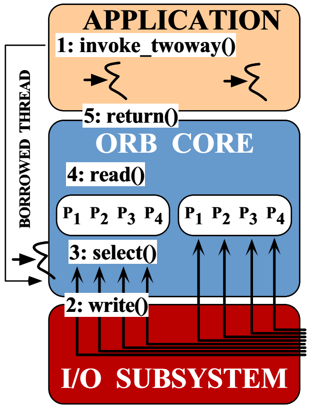
- A thread dedicates to handle I/O
- Extra context switch between layers
- Use GIOP sequence number to demultiplex replies
- Priority inversion — solution: prioritize queues

Leader/Follwer Connection – Client Side



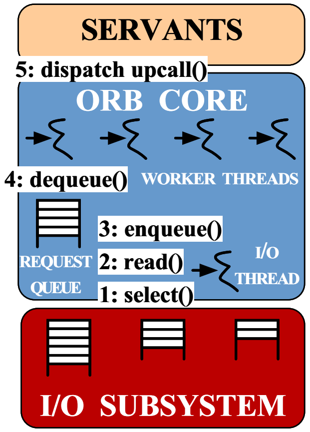
- Reduced context switch (limited)
- More complex to implement
- Locking overhead may outweigh performance gain from saved context switching
- Priority inversion possible if leader disrespect the priority information

Non-multiplexed Connection – Client Side



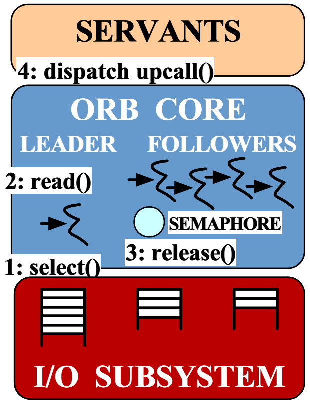
- Pre-established connections for various priorities
- No resource contention — no priority inversion, locking overhead
- Non-scalable

Worker Thread Pool – Server Side



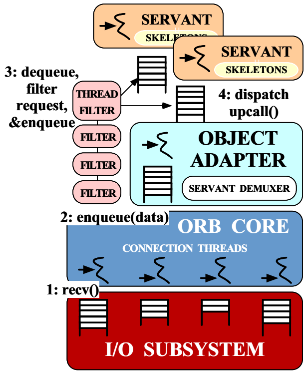
- A dedicate I/O thread
- Straightforward producer/consumer design
- Excessive context switching and synchronization
- Priority inversion caused by queueing and connection multiplexing

Leader/Follwer Thread Pool – Server Side



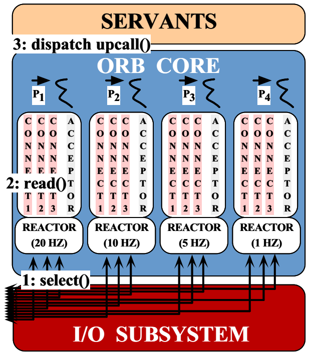
- Each thread handles a complete request
- Minimize context switching
- Priority inversion by connection multiplexing

Threading Framework – Server Side



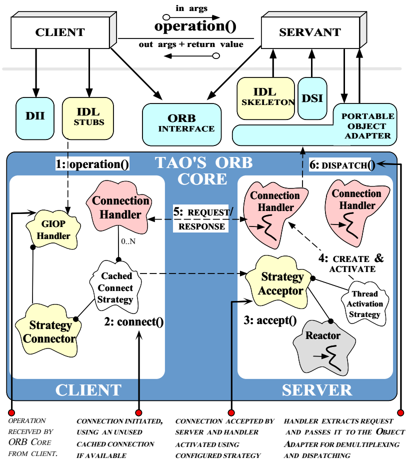
- Application installable filters allow intercepting, modifying, examining requests, and altering ORB behavior
- Priority inversion may occur in the filter chain
- Over generality leads to excessive context switching and synchronization overhead

Reactor-per-Thread-Priority – Server Side



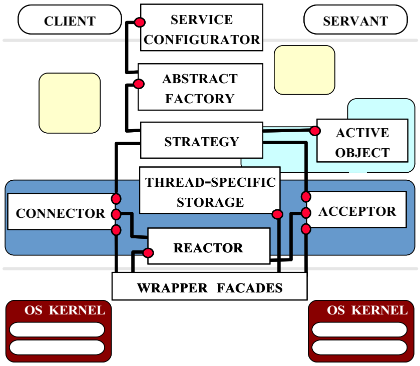
- Integrate endpoint demultiplexing and dispatching
- Minimize priority inversion and non-determinism
- Reduce context switching and synchronization overhead
- Non-scalable
- Can associate each reactor with a thread pool remove serialized service in a priority group

Architecture of TAO ORB Core



- Thread-per-connection
- Reactive
- Thread pool (future)
- Resource management
- Connection management

Class Collaboration in TAO



Patterns used in TAO

- Factories produce strategies
- Strategies implement interchangable policies
- Service Configurator permits dynamic configuration
- Concurrency strategies implemented using Reactor, Active Object, etc
- Connector/Acceptor decouple transport type from operations