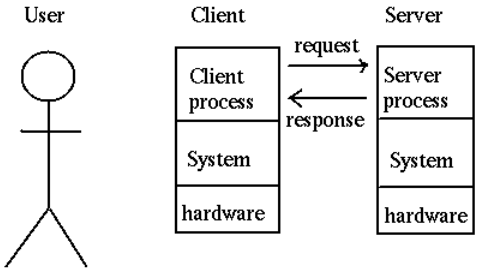
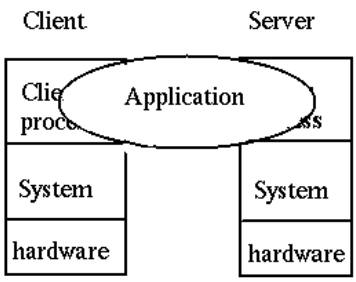
Client-Server Applications

Prof. Sanjeev Setia
Distributed Software Systems
CS 707

Client Server Systems



Client/Server Application



Overview

- Common communication patterns in distributed applications
  - Client-Server
  - Group (Multicast)
  - Function-shipping/Applets
- Client: process that requests service
- Server: process that provides service
- Client usually blocks until server responds

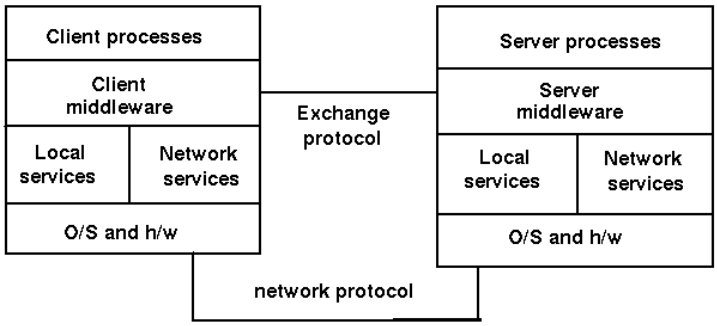
Overview cont'd

- Client usually invoked by end users when they require service
- Server usually waits for incoming requests
- Server can have many clients making concurrent requests
- Server usually a program with special privileges

Client and Server Functions

- Clients
  - interacts with users through a user interface
  - performs application functions
  - interacts with client *middleware* using middleware API
  - receives response and displays it if needed
- Servers
  - implement services
  - invoked by server *middleware*
  - provide error-recovery and failure-handling services

Middleware



Middleware

- Definitions
  - Middleware is a set of common business-unaware services that enable applications and end-users to interact with each other across a network
  - distributed system services that have standard programming interfaces and protocols ... services “sit in the middle” above OS and network software and below industry-specific applications
  - the “/” in client/server applications
  - software nobody wants to pay for

Examples

- ftp, email
- Web browsers
- Database drivers and gateways
- OSF's DCE (Distributed Computing Environment)
- OMG's CORBA (Common Object Request Broker Architecture)

Functional View of Middleware

- Information exchange services
- Application-specific services
  - specialized services, e.g. transactional services and replication services for distributed databases, groupware services for collaborative applications, specialized services for multimedia applications
  - *business-unaware*
- Management and support services
  - needed for locating distributed resources and administering resources across the network

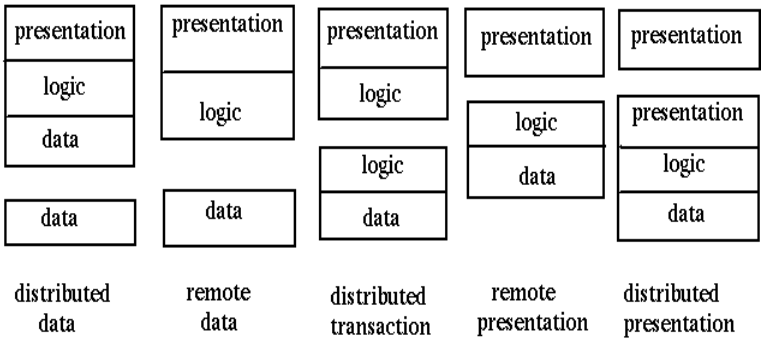
Commercial Middleware

- Middleware components that provide only one service
  - HTTP for retrieving remote documents, SUNRPC for RPC, etc.
- Middleware environments that combine many services
  - Integrates RPC, security, directory, time and file services
  - DCE, CORBA, Microsoft DCOM, .NET, Java
- Compound middleware environments that combine many middleware environments into a single framework, e.g. transaction management + RPC/RMI

Application Software Architectures

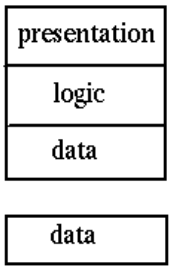
- Many applications can be considered to be made up of three software components or logical tiers
  - user interface
  - processing layer
  - data layer
- Client/server architectures
  - single-physical tiered, two-physical tiered
  - multi-tiered

“Gartner Group” Configurations



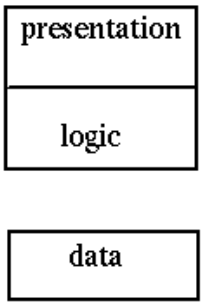
Distributed Data

Example: Distributed Database



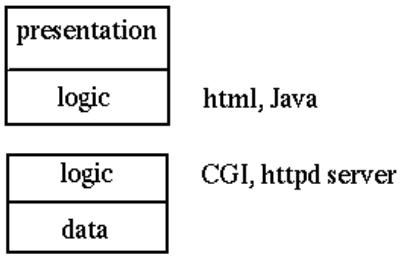
Remote Data

Example: Network File Systems



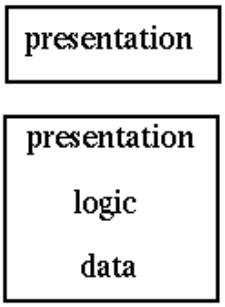
Distributed Programs

Example: World Wide Web



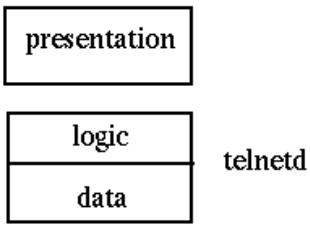
Distributed Presentation

Example: X Windows

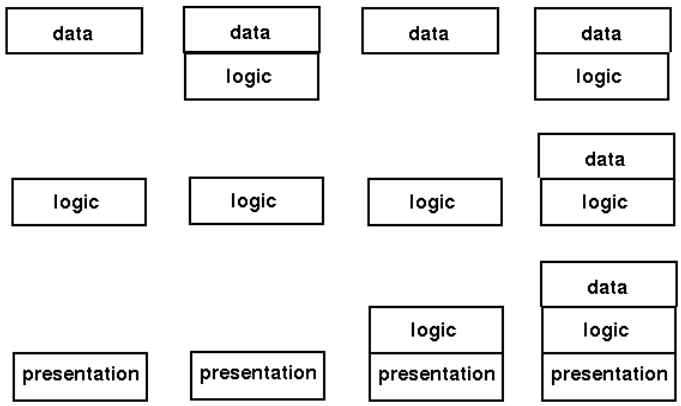


Remote Presentation

Example: telnet



Three-tier architectures



Motivation for multi-tier architectures

- Frees clients from dependencies on the exact implementation of the database
- It allows “business logic” to be concentrated in one place
  - Software updates are restricted to middle layer
- Performance improvements possible by batching requests from many clients to the database
- Database and business logic tiers could be implemented by multiple servers for scalability

Fat vs thin clients

- Thin client = network computer
  - Typically no local storage
- Fat client = typical desktop PC, workstation
- Motivation for thin clients: hidden costs of system administration and support
  - Network computers a move towards *centralized* system admin but *local* processing at client
  - Java (mobile code) an enabling technology
- Degrees of “thinness”, e.g. PDAs

Issues in Client design

- Must know or find out the location of the server
- Which protocol to use: reliable or unreliable?
- Blocking (synchronous) request or non-blocking (asynchronous)

Issues in Server Design

- Connection-oriented or connection-less servers
  - TCP or UDP?
- Concurrent or iterative servers: handle multiple requests concurrently or one after the other?
- Stateful or stateless servers
- Multi-protocol, multi-service servers

Connection-less vs connection-oriented servers

- protocol used determines level of reliability
- TCP provides reliable-data delivery
  - verifies that data arrives at other end, retransmits segments that don't
  - checks that data is not corrupted along the way
  - makes sure data arrives in order
  - eliminates duplicate packets
  - provides flow control to make sure sender does not send data faster than receiver can consume it
  - informs both client and server if underlying network becomes inoperable

Connection-less servers

- UDP unreliable – best effort delivery
- UDP relies on application to take whatever actions are necessary for reliability
- UDP used if
  - application protocol designed to handle reliability and delivery errors in an application-specific manner, e.g. audio and video on the internet
  - overhead of TCP connections too much for application
  - multicast

Stateful vs stateless servers

- State ≡ Information that server maintains about the status of ongoing interactions with clients
- Stateful servers
  - state information can help server in performing request faster
  - state information needs to be preserved across (or reconstructed after) crashes
- Stateless servers
  - quicker and more reliable recovery after crashes
  - smaller memory requirements
- Stateless servers: application protocol should have ***idempotent*** operations

Concurrency in servers

- Concurrency needed if several clients and service is expensive
- Operating system support
  - Multiple processes
  - Threads
  - Asynchronous I/O, e.g. using select() system call
- Process/thread preallocation for improving performance
- Delayed process/thread allocation