Computer Science Program, The University of Texas, Dallas

Client-Server Architecture

Clients and Servers

Client/Server with File Servers

Client/Server with Database Servers

Client/Server Communication

Client/Server with Transaction Processing

Client/Server Groupware

Web Client/Server

Paradigm Shift: Past, Present and Future

- **Centralized processing**

  A host computer (often a mainframe) handls all processing, including input, output, data storage and retrieval

  ☛ *Predominant computing mode in the '70s*

- **Distributed processing**

  A number of computers (minis, workstations, PCs, ...) handle all processing, They are distributed physically and connected thru a communications network

  ☛ *The computing mode of the present*

- **Cooperative processing**

  A number of computers (minis, workstations, PCs, ...) handle all processing, They are distributed physically and connected thru a communications network

  Processing thru sharing of resources, transparently to the users

  ☛ *The computing mode of the future*

Clients and Servers

Basic Definition

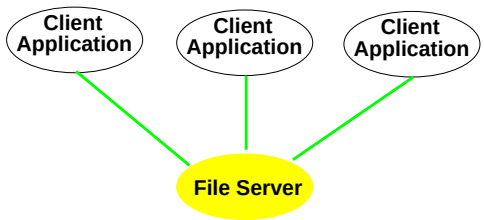
- Client: requests for services
- Server: provides services
- Service: any resource
  (e.g., data, type defn, file, control, object, CPU time, display device, etc.)

Typical Properties

- A service request is about "what" is needed, and it often made abstractly
  It is up to the server to determine how to get the job done
  -> the notion of module
  (cf. MILs, ADTs --- Larch, Z)
- The ideal client/server software is independent of hardware or OS platform
- The location of clients and servers are usually transparent to the user
- A client may become a server; a server may become a client
- A client/server system can be scaled
  horizontally, i.e., by adding/removing client workstations
  with only a slight performance impact
  vertically, i.e., by migrating to a larger and faster server machines
  or multiservers

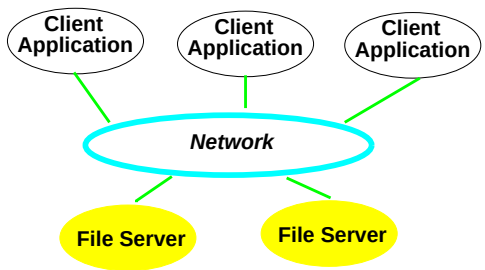
Client/Server with File Servers

Centralized

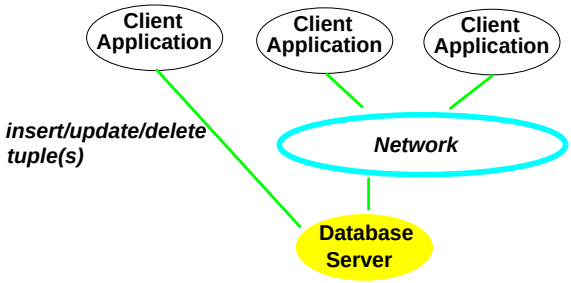


- *The client passes requests to the file server (software) for file records*
- *Clients can reside in the same machine or separate machines (typically PCs)*
- *Requests can be either local or over a network*
- *Indispensable for documents, images, drawings, and other large data objects*

Distributed



Client/Server with Database Servers



- *At present the majority of existing client/server-based software is to be found in the area of databases, and it is here that the greatest challenge to any corporation currently lies.*
  [Richard Finkelstein, President, Performance Computing]

- Events (violation of integrity constraints, temporal conditions, errors) trigger event handlers
  -> implicit invocation, blackboards, events

- A DBMS also offers features for recovery and concurrency control
  Oracle, sybase, Informix, Ingres, Gupta, ...

Client/Server Communication

Sockets -> RPCs -> MOMs and ORBs

Sockets

- introduced in 1981 as the Unix BSD 4.2 generic interface
- provide Unix-to-Unix communications over networks
- The Windows socket API (Winsock): based on the Unix BSD 4.2 sockets interface

Net_id.Host_id Internet Address	Port Address

unique Net_id, multiple Host_id, Ports (entry points to applns on a host)

- Principal transport service calls

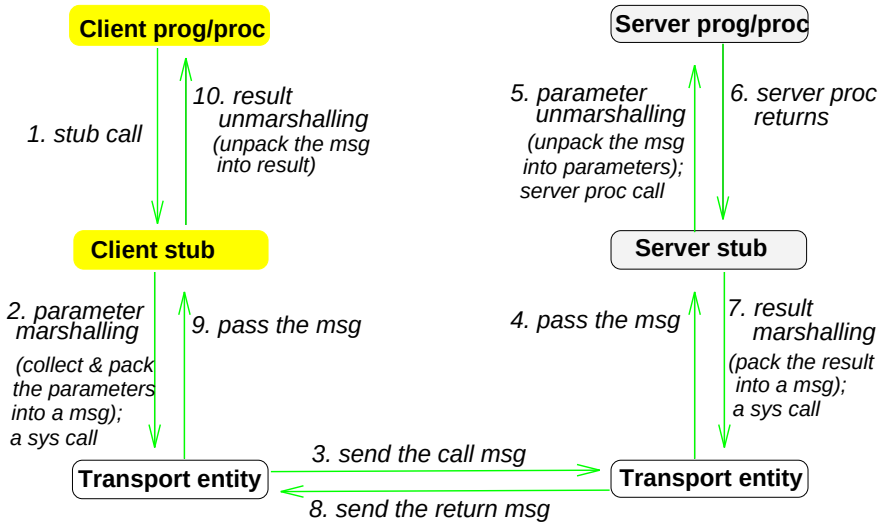
Create a socket
Socket (addr format, type: connection-oriented/connectionless, protocol:TCP/IP/...)
Associate an Ascii name to a previously created socket
Bind(...)
Create a queue for storing incoming connection requests (blocking/non-blocking)
Listen(...)
Remove a connection request from the queue or wait for one (blocking/non-)
Accept(...)
Initiate a connection with a remote socket
Connect(local_socket, remote_socket)
Terminate the connection on a socket
Shutdown(local_socket, remote_socket)
Send a message thru a goven socket
Send(local_socket, remote_socket, buffer, bytes)
Receive a message thru a goven socket
Recv(local_socket, remote_socket, buffer, bytes)
Check a set of sockets to see if any can be read or written (blocking)
Select(...)

Client/Server Communication

Sockets -> RPCs -> MOMs and ORBs

RPCs (Remote Procedure Calls)

- a transparent mechanism to give the client procedure the illusion that it is making a direct call on the distant server procedure
- ***stubs:***
  local procedures (e.g., read(file_id, buffer, count)) hiding details of network comm.
- 10 steps to execute a RPC

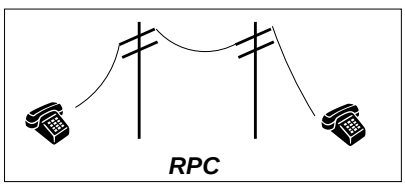


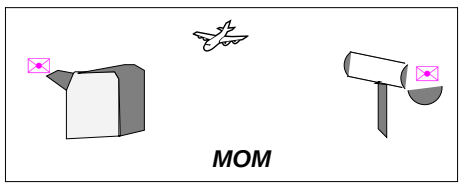
Client/Server Communication

Sockets -> RPCs -> MOMs and ORBs

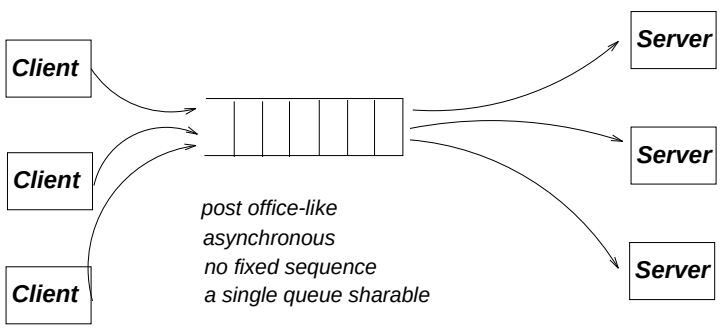
MOM (Message-Oriented Middleware)

Every DAD (Distributed Application Development) needs a MOM

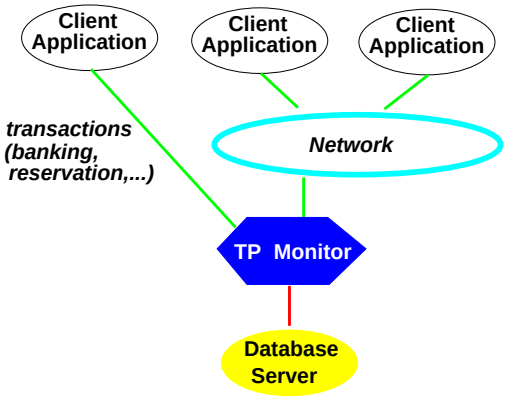




- *Many-to-many messaging via queues*



Client/Server with Transaction Processing



- **Transaction changeAddress (p: String, newAddress: Address)**

```
begin
        oldAddress <- retrieve address from Person where "name = p"
        if oldAddress = newAddress then return error1
                                   else update Person where "p = name"
                                                with address <- newAddress
end
```

- **Transaction increaseSalary (increment: Dollar)**

```
begin
end   ...
```


Client/Server with Transaction Processing

- *Transactions are a way to make ACID operations a general commodity*
  [Transaction Processing Concepts and Techniques, Jim Gray and Andreas Reuter, 1993]

❒ ***Atomicity***
- ❑ a transaction is an indivisible unit of work
- ❑ an all-or-nothing proposition
- ❑ all updates to a database, displays on the clients' screens, message queues
- ❑ e.g., salary increase for all 1 million employees or none

❒ ***Consistency***
- ❑ a transaction is an indivisible unit of work
- ❑ S -> [T | abort] -> S
- ❑ integrity constraints (e.g., mgr.salaray > salary)

❒ ***Isolation***
- ❑ a transaction's behavior not affected by other transactions running concurrently
- ❑ e.g., reserve a seat
- ❑ serialization techniques

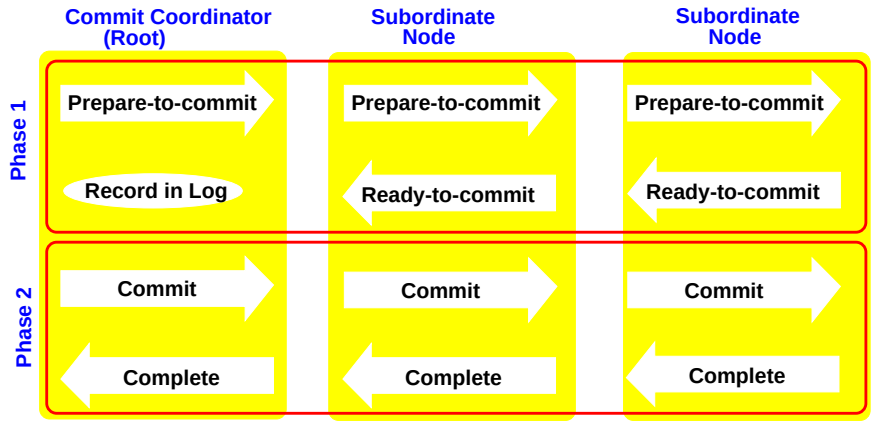
❒ ***Durability***
- ❑ persistence
- ❑ a transaction's effects are permanent after it commits.

ACID is like motherhood and apple pie. It's necessary- and you can't have too much of it. OK, enough preaching.

Client/Server with Transaction Processing

Two-Phase Commit Protocol

- *To synchronize updates on different machines so that they either all fail or all succeed*
- *centralize the decision to commit but giving each participant the right of veto*
- *like a Christian marriage*



- If the subordinates have spawned pieces of the transaction on other nodes, they must propagate the prepare-to-commit-command in the first phase
  -> a transaction tree
- If any of the participants returns a transaction failure, the root node tells all its subordinates to perform a rollback

Client/Server Groupware

Why?

support for business reengineering: bcfh -> maximize profit
change the way people communicate with each other
helps manage (and track) the product thru its various phases
computer-supported cooperative work (CSCW)
collaborative/workgroup computing
return on investment: 16% to 1666% on a median investment of $100K
[a study of 65 Notes users in 1994]

Five foundation technologies

"combining technologies and creating new synergy"

- **Multimedia document management**
  from electronic imaging (scanning, digitization, display, storage and retrieval)
  to document (component types: text, image, graphics, faxes, mail, voice clips, BBs)
- **Workflow**
- **E-mail**
- **Conferencing**
  thru microphones, video windows, electronic whiteboards, controlled access to shared document
- **Scheduling**
  electronic scheduling of meetings, sharing calendars and "to do" lists, triggering events

Why not just DBMS?

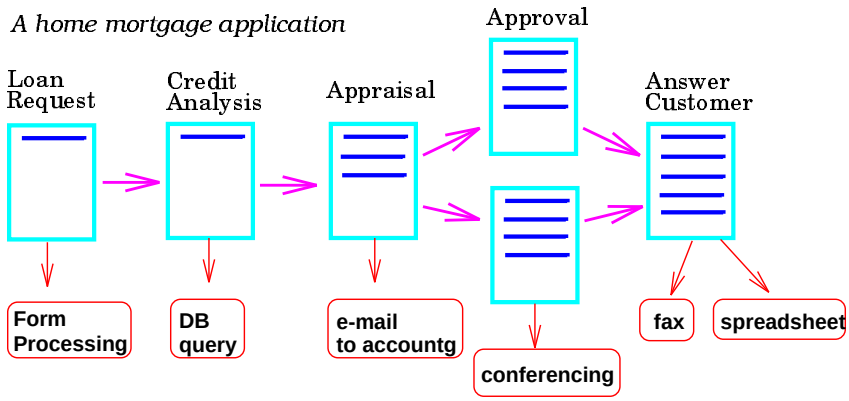
DBMS: highly structured data
Groupware: highly unstructured data document database

Why not just TP Monitors?

TP Monitors: transaction processes
Groupware: (currently) not transaction-oriented in the ACID sense

Client/Server Groupware

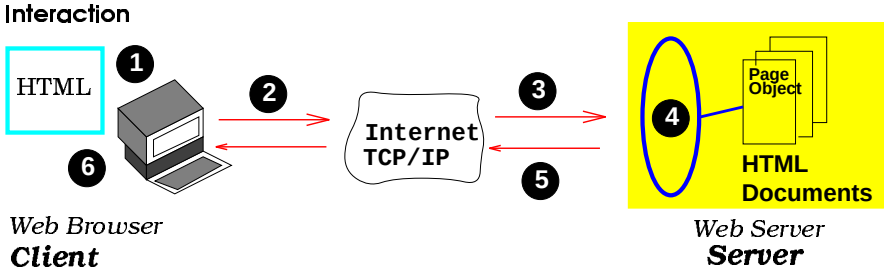
Workflow



- Many office workers spend a high percentage of their time processing documents
- The "workflow river" carries the flow of work from port to port, value being added along the way
- The workflow definition must take into consideration:
  - *Routes* along which the object moves
  - *Rules* about what information is routed and when
    E.g., "If the loan is over $100, 000, send it to the supervisor within the next hour or else send it to the next hop"
  - *Roles* define job functions independently of the people who do it
    E.g., the "supervisor" role can be handled by users "Adam" and "Eve"

Web Client/Server

Interaction



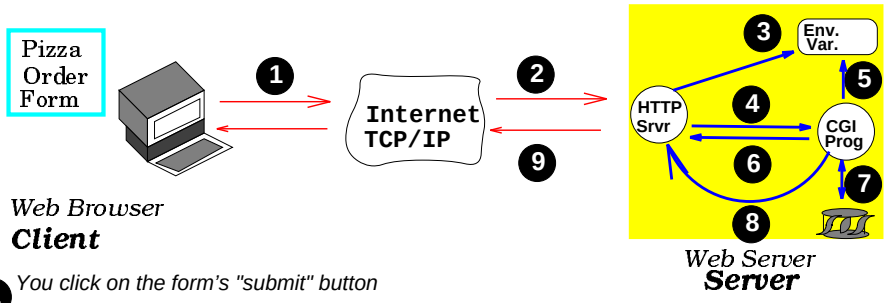
1. *You select a target URL (Univeral Resource Locator - platform-independent naming scheme)*

http	://www.utdallas.edu	:80	:~ttsx19c/java/toys.html
Protocol	*Server Address*	*Port Number*	*Target Resource Path*

2. *Brower takes the URL, embeds it inside an HTTP request, and sends it to the server*
   - *HTTP (HyperText Transfer Protocol), a stateless RPC, is used to retrieve URL-named resources 1) establishes a client/server connection, 2) transmits and receives parameters including a returned file, and 3) breaks the client/server connection*
   - ***STATELESS:** a new connection for each user request - simple but inefficient*
3. *The HTTP server receives the reqest on the port, makes a socket connection,*
4. *The server finds the requested HTML file (Page Object) and concatenates it with status info*
   - *HTML (HyperText Markup Language) describes the structure of a Web document, provide font and graphics info, and define hyperlinks to other Web pages & Internet resources)*
5. *The server ships back the file, and closes the connection*
6. *The Browser interprets the HTML command and displays the page content and invokes a helper application (e.g., xview, audiotool)*

Web Client/Server

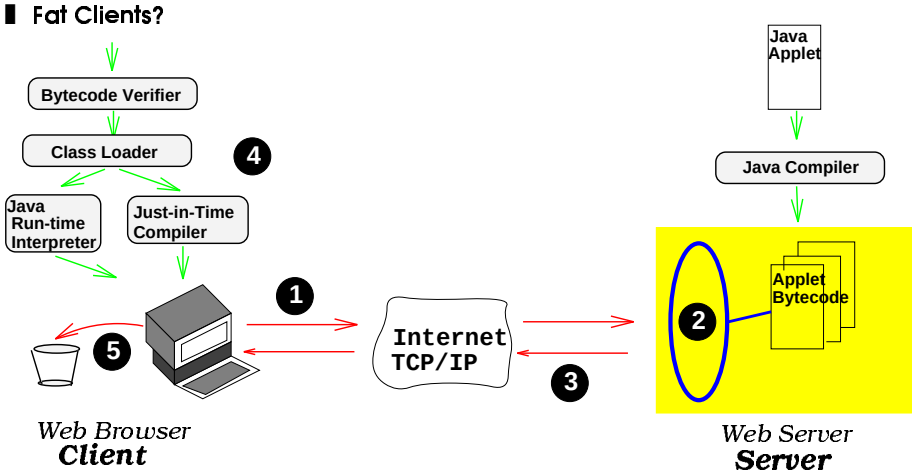
- Forms, database updates, (primitive) electronic commerce



1. *You click on the form's "submit" button*
   The Web browser collects the data within the form, assembles it into a string of name/value pairs, specifies a POST method, the URL of the target program in the "cgi-bin" directory
2. *The HTTP server receives the method invocation via a socket connection*
3. *The server parses the message to discover that it's a POST for the "cgi-bin" program*
   -> start a CGI interaction:
   The HTTP server sets up the environment variable (e.g., content_length), in a blackboard
4. *The HTTP server starts a CGI program*
5. *The CGI program reads the environment variables*
6. *The CGI program receives the message body via the input pipe*
7. *The CGI program does some work, typically by interacting with a DBMS, TP Monitor*
8. *The CGI program returns the results via the output pipe*
9. *The HTTP server returns the results to the Web brower*

Web Client/Server

- Fat Clients?



1. *A Web browser requests a Java applet*
   The Browser initiates a separate TCP/IP session to download each applet within a Web page
2. *The server retrieves the requested applet*
3. *The server sends the applet*
4. *The browser loads the applet into the client's memory and executes it*
   A poor man's compound document architecture
   The applet's region does NOT visually integrate with the rest of the page
5. *The browser deletes the applet from memory when it exits the Web page*