
Client-Server Style

Jing Dong

CS6362 Software Architecture and Design

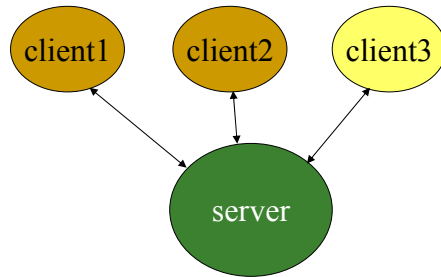
1

Client-Server Style

- Suitable for applications that involve distributed data and processing across a range of components.
- Components:
 - **Servers**: Stand-alone components that provide specific services such as printing, data management, etc.
 - **Clients**: Components that call on the services provided by servers.
- Connector: The network, which allows clients to access remote servers.

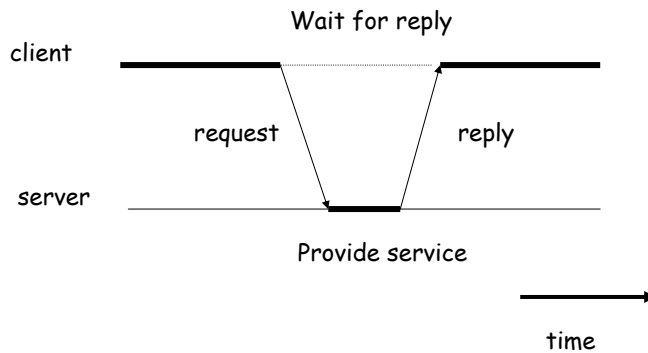
2

Client-Server Style



3

Client-Server Style



4

Different Kinds of Servers

- A stateful server maintains state information about each communication session.
- A stateless server treats each request as a separate one.
- A concurrent server can serve multiple requests at the same time
- An iterative server serves one request at each time.

5

Paradigm Shift: Past, Present and Future

- Centralized processing (70's)
 - A host computer (often a mainframe) handles all processing, including input, output, data storage and retrieval
- Distributed processing (present)
 - A number of computers (minis, workstations, PCs, ...) handle all processing. They are distributed physically and connected through a communication network
- Cooperative processing (future)
 - A number of computers (minis, workstations, PCs, ...) handle all processing. They are distributed physically and connected through a communication network
 - Processing through sharing of resources, transparently to the users

6

Clients and Servers

- Basic Definition
 - Server: provides services
 - Client: requests for services
 - Service: any resource (e.g., data, type definition, file, control, object, CPU time, display device, etc.)
- Typical Properties
 - A service request is about "what" is needed, and it is often made abstractly. It is up to the server to determine how to get the job done
 - The locations of servers are usually transparent to the user
 - A client may become a server; a server may become a client
 - The ideal client/server software is independent of hardware or OS platform
 - 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

7

Client-Server Style Examples

- File servers
- Database servers
- Object servers

8

Client-Server Style Examples

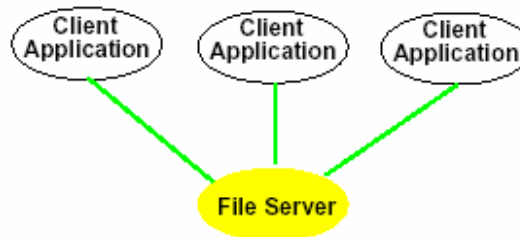
■ File Servers:

- Primitive form of data service.
- Useful for sharing files across a network.
- The client passes request for files over the network to the file server.
- Clients can reside in the same machine or separate machines (typically PCs)
- The client passes requests to the file server (software) for file records
- Requests can be either local or over a network
- Indispensable for documents, images, drawings, and other large data objects

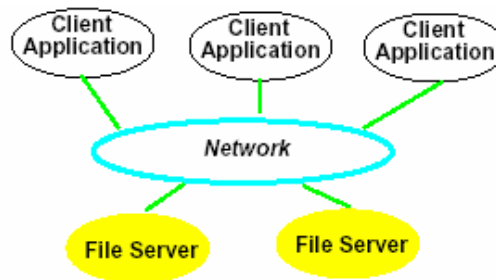
9

Client-Server Style Examples

Centralized



Distributed



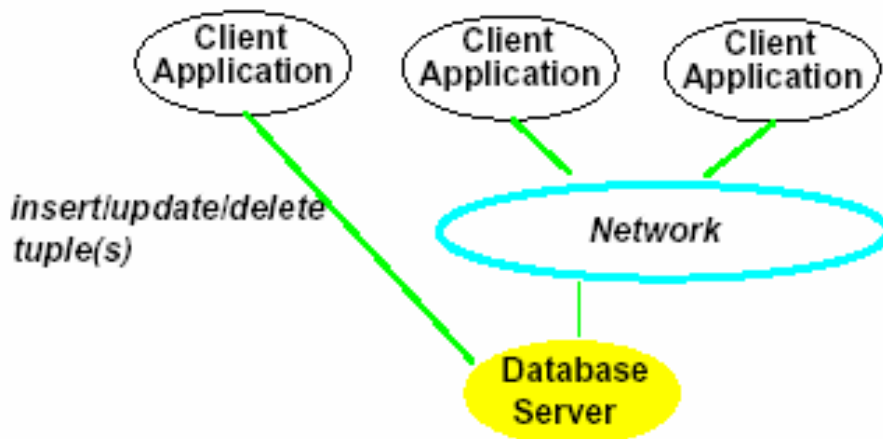
10

Client-Server Style Examples

- **Database Servers:**
 - More efficient use of distributing power than file servers.
 - Client passes SQL requests as messages to the DB server; results are returned over the network to the client.
 - Query processing done by the server.
 - No need for large data transfers.
 - Transaction DB servers also available.
- 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.
- A DBMS also offers features for recovery and concurrency control

11

Client-Server Style Examples



12

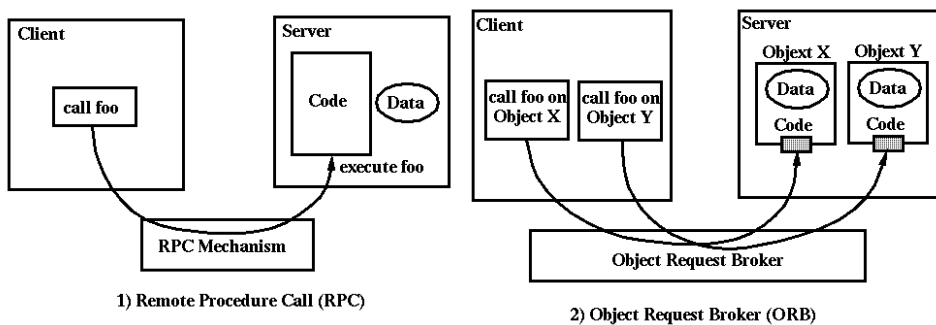
Client-Server Style Examples

■ Object Servers:

- ❑ Objects work together across machine and network boundaries.
- ❑ Object Request Brokers (ORBs) allow objects to communicate with each other across the network.
- ❑ Interface Definition Languages (IDLs) define interfaces of objects that communicate via the ORB.
- ❑ ORBs are the evolution of the RPC.

13

Client-Server Style Examples



14

IPC

- Inter-Process Communications
 - How processes will communicate and synchronize with one-another.
 - communications mechanisms:
 - shared memory
 - very fast
 - can't use over a network
 - message passing
 - can use over a network
 - slower
 - will consider only message passing (most important)

15

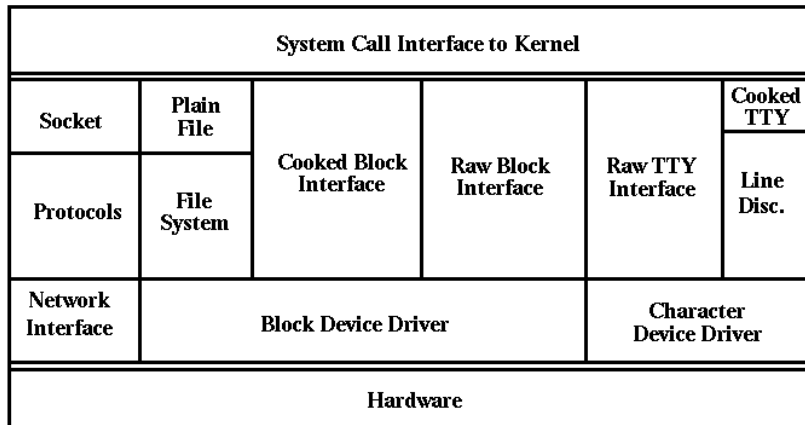
IPC Protocols

- Basic message-passing mechanisms provide for a byte-stream only.
- Must implement various protocols on top of this
 - Sockets
 - RPC (remote procedure call)
 - DO (distributed objects)

16

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



17

Sockets code example

```

public class Server {
    public static void main(String[] args) throws Exception {
        ServerSocket server = new ServerSocket(1234);
        Socket client = server.accept();
        BufferedReader fromClient = new BufferedReader(
            new InputStreamReader(client.getInputStream()));
        System.out.println(fromClient.readLine());
    }
}

public class Client {
    public static void main(String[] args) throws Exception {
        Socket server = new Socket("search", 1234);
        DataOutputStream toServer = new DataOutputStream(
            server.getOutputStream());
        toServer.writeBytes("hello server");
        server.close();
    }
}

```

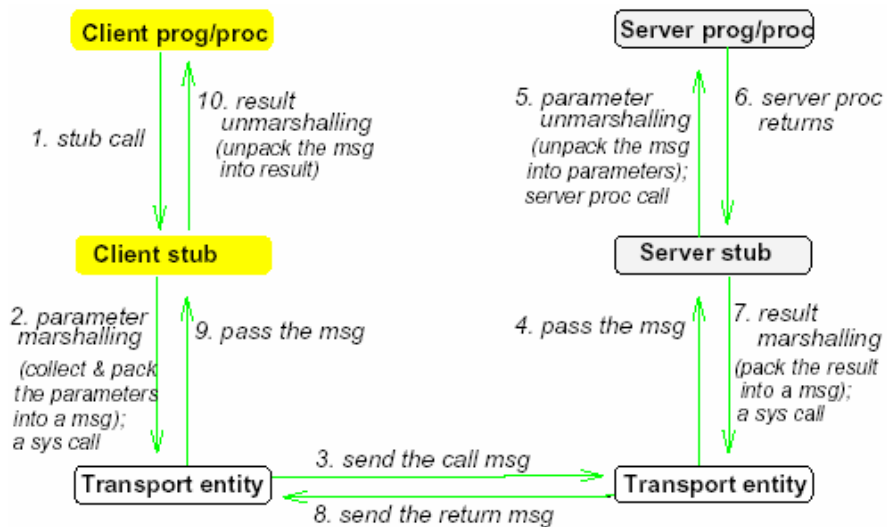
18

RPC

- 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 communication
 - 10 steps to execute a RPC

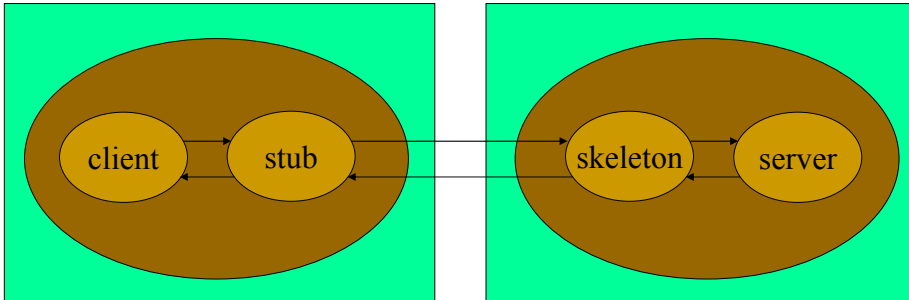
19

RPC



20

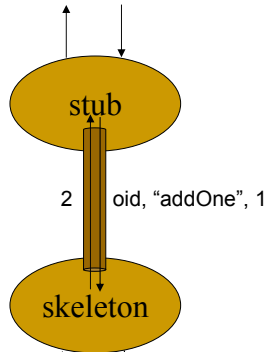
Distributed Objects



21

Marshalling Parameters

```
int result = arithmeticServer.addOne(1)
```



```
public int addOne(int x) { return x+1; }
```

22

Client-Server Advantages

- Distribution of data is straightforward
- transparency of location
- mix and match heterogeneous platforms
- easy to add new servers or upgrade existing servers
- maintainability — both servers and clients can be changed “independently” (both hardware and software), server is in one place
- performance — processors can be dedicated to specific tasks (e.g., client processor handles user interface with special purpose graphics hardware)
- scalability — adding clients is low-cost (up to a limit), changing (or adding) servers can be relatively transparent

23

Client-Server Disadvantages

- No central register of names and services -- it may be hard to find out what services are available
- Server is bottleneck
- Protocol must be compatible

24