CS323 Operating Systems
Multi-Computers/Distributed Systems

Yuanyuan Zhou

Lecture 34

4/23/2003

Content of this lecture

- Reminder
  - Conflict exam: Wed 5:30-6:30pm, DCL 2501
  - Doctor's notes
- Distributed Systems vs. Network OS
- RPC vs. RMI
- Multi-processor (read textbook)

Distributed vs. Network Operating System

- **Network Operating System (NOS):** a collection of software and associated protocols that allows a set of autonomous computers, which are interconnected by a computer network, to be used together in a convenient and cost-effective manner. Each host runs its own non-network operating system; the network is controlled by user programs running on each host.
  - Typically used to interconnect large, architecturally diverse, and geographically dispersed autonomous systems.
  - Analogy: United Nation
- **Distributed Operating System:** A single, homogeneous operating system controls all hosts on the network, and the network itself; each host does not have an individual operating system of its own. The hosts are not ``autonomous'', as in an NOS.
  - Typically used for local networks of mini- and micro-computers.
  - Analogy: Federated Governement

Network Operating Systems

- Allow users to access the various resources (e.g., programs and data) on each network host.
- Limit access to authorized users of each particular resource.
- Make the network and the eccentricities of the host computers transparent to the users.
- Make the use of remote resources appear to be identical to the use of local resources.
- Provide uniform accounting procedures throughout the network.
- Provide current network documentation on-line.
- Provide more reliable operation than would be possible on a single host, especially over a group of equivalent hosts.

Distributed-Operating System

- Users access remote resources as if they were local resources
  - location transparency
- Data Migration
  - data moved to system needing it
- Computation Migration
  - program moved to system with appropriate data
- Process Migration
  - a process moves from one machine to another
- Job Migration
  - Job moves so as to balance load.
- Control execution of multi-step jobs in which the several steps can be executed on different hosts.
- Balance network load by moving jobs to underutilized hosts (assuming equivalent hosts).
- Move jobs to the host best suited to each task (assuming non-equivalent hosts).

Distributed Communication - Sockets

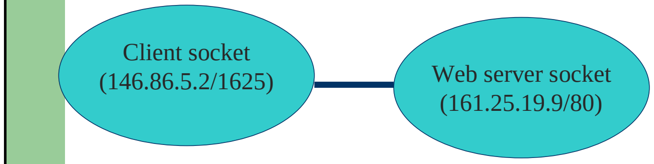
- **Socket** - endpoint for communication. A pair of processes or threads communicating over a network employs a pair of sockets.
- Socket is made up of
  - IP address
  - port number.
- Examples
  - telnet server listens on port 23
  - ftp server listens on port 21
  - web server listens on port 80.
- All ports below 1024 are well-known. Other applications can assign a port with an arbitrary number larger than 1024.
- All connections must be unique, i.e., if another process on the same host wishes another connection with the same web server, it must be assigned a new port number larger than 1024. This ensures that all connections consist of a unique pair of sockets.

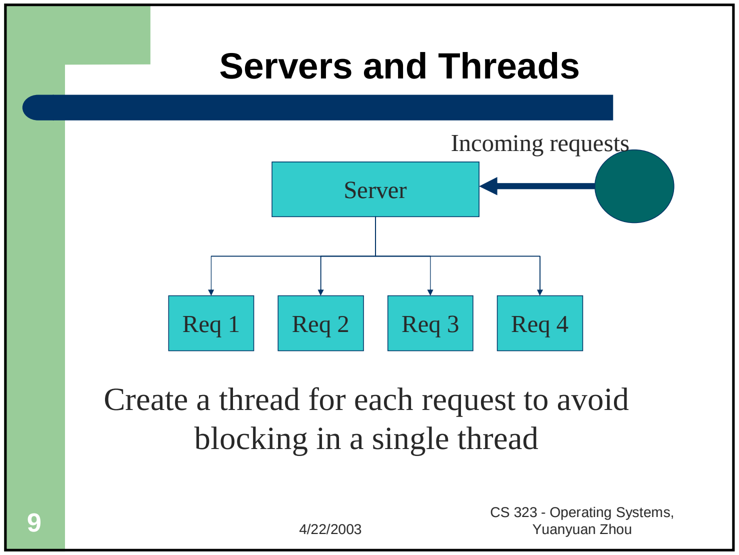
Java Sockets

- Three different types of sockets:
  - connection-oriented (TCP) sockets
    - implemented with the Socket class
  - connection-less (UDP) sockets
    - implemented with the DatagramSocket class
  - multicast socket
    - implemented as MulticastSocket class

Socket Communication

- Sockets use a client-server architecture.
  - The server waits for an incoming client requests by listening to a specified port.
  - Once a request is received, the server accepts a connection from the client socket to complete the connection.





Servers and Threads

Incoming requests

Server

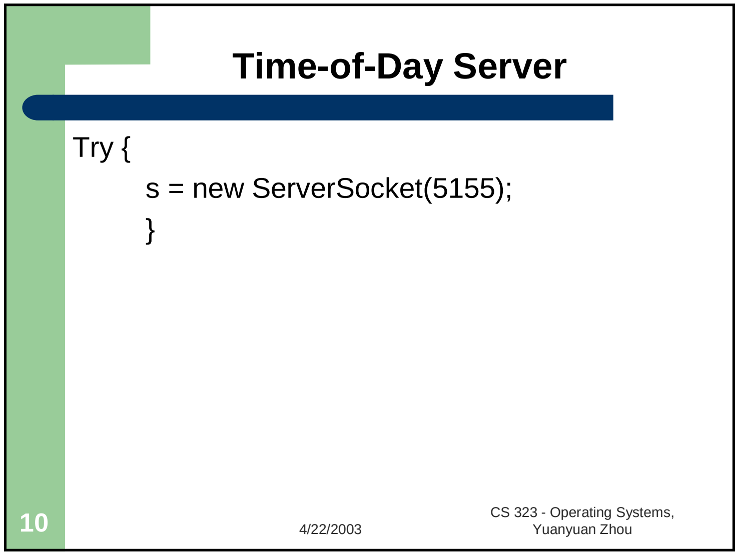
Req 1

Req 2

Req 3

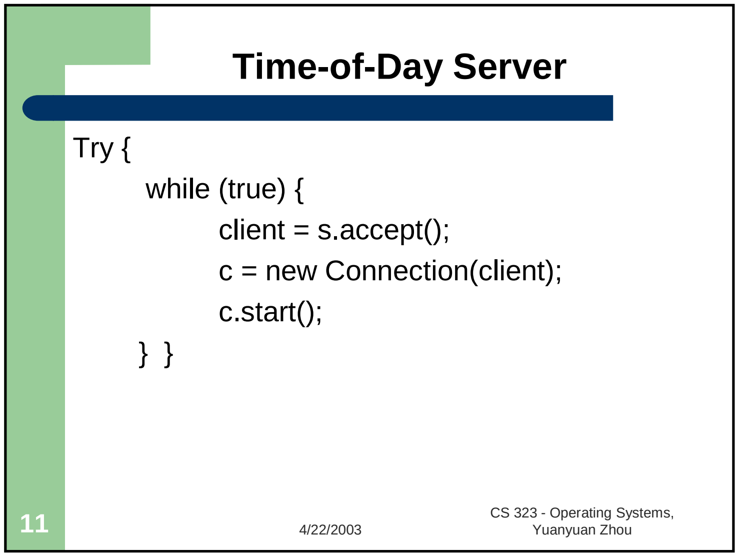
Req 4

Create a thread for each request to avoid blocking in a single thread



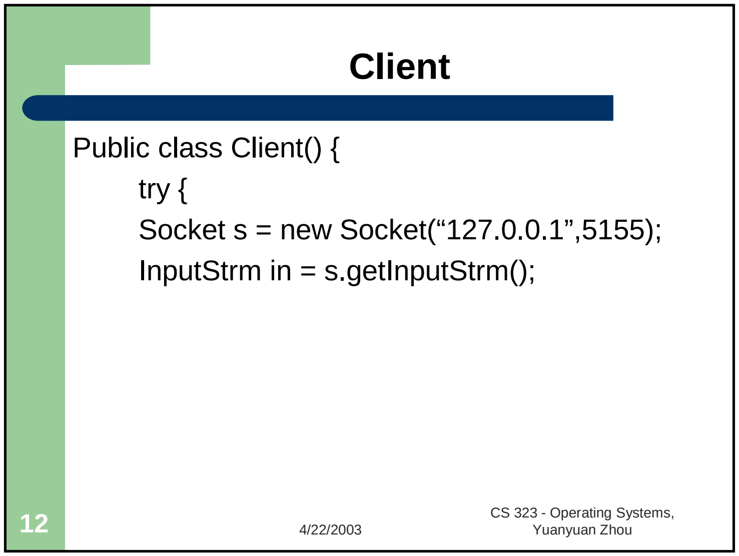
Time-of-Day Server

```
Try {
        s = new ServerSocket(5155);
        }
```



Time-of-Day Server

```
Try {
        while (true) {
                client = s.accept();
                c = new Connection(client);
                c.start();
        } }
```



Client

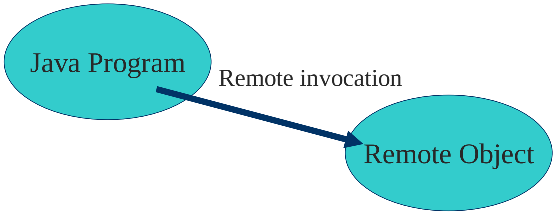
```
Public class Client() {
        try {
        Socket s = new Socket(“127.0.0.1”,5155);
        InputStrm in = s.getInputStrm();
```

Remote Procedure Call

- RPC
  - Parameters of RPC are marshaled into message
  - Message sent and local process waits
  - Remote machine receives message, spawns remote process
  - Remote process assumes same protection domain as local process
  - Remote process assembles parameters and makes procedure call
  - Remote process marshals return parameters into message
  - Message sent and remote process dies
  - Local process resumes and unpacks result
- Implementation: Client program must be bound with a small library procedure, called **client stub** that represents the server procedure in the client's address space. Similarly, the server is bound with a procedure, called the **server stub**. These procedures hide the fact that the procedure call from client to server is not local.

Java Remote Method Invocation

- RPC with objects
- Includes objects as parameters



RMI

- Client has stub for remote object – proxy for the remote object
  - Parcel - marshalling
- Server has skeleton
  - Unmarshals parameters
  - Invokes method on remote machine

Parameter Passing Behavior

- Local objects – object serialization (passed by copy)
- Remote objects passed by reference
- Local objects must implement java.io.Serializable

Finding Objects

- Rmiregistry
  - Remote object registers using Naming.rebind()
  - Client obtains reference to a remote object using Naming.lookup()

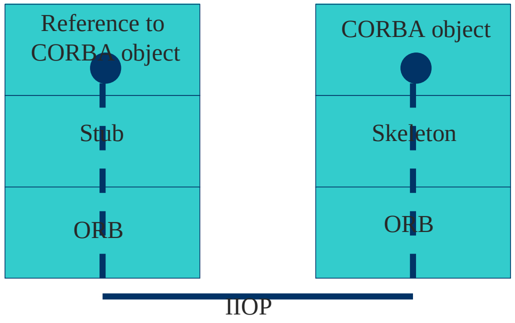
Discussion

- Advantage and disadvantage of RMI

Shared Object-Based Middleware

- Desire to distribute systems written in C, C++, and other languages.
- **CORBA** (Common Object Request Broker Architecture)
  - Convenient mechanism to allow object communication over heterogeneous platforms
  - allows applications written in different languages to communicate
- **Interface Definition Language (IDL)**
  - allows a distributed object to describe an interface to the services provided
  - Independent of any specific programming languages
  - The client obtains a reference for the remote object, and communicates with the interface specified with the IDL.
- **Object Request Broker (ORB)**.
  - the client' stub communicates with the client's ORB. ORBs at the client and server communicate via the **Internet InterORB Protocol (IIOP)**.
  - The server ORB allows the server to register new CORBA objects and it is responsible for accepting requests from client ORBs.
  - The server's skeleton is called, once the request arrives and the implementation of the remote method on the server is invoked.

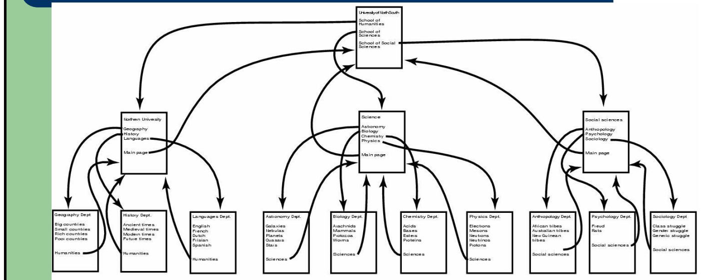
CORBA MODEL



Distributed Shared Memory

- What is DSM?
  - Provide shared memory to applications on a cluster of machines each with their own memory
- How?
  - When a CPU does a LOAD or STORE on a page not in memory, trap to OS occurs.
  - The OS locates the page, and asks the CPU currently holding it to unmap the page and send it over the interconnection network.
  - When it arrives, the page is mapped in and the faulting instruction is restarted.
- DSM might use large page sizes.
  - Advantage: prefetching effect
  - Disadvantage: Too large page sizes can cause **false sharing**.
- Consistency: sequential vs. relaxed

Document-Based Middleware (1)



- The Web
  - a big directed graph of documents

Document-Based Middleware (2)

- How the browser gets a page
  - Asks DNS for IP address
  - DNS replies with IP address
  - Browser makes connection
  - Sends request for specified page
  - Server sends file
  - TCP connection released
  - Browser displays text
  - Browser fetches, displays images

Coordination-based Middleware

- **Linda System**: Independent processes communicate via abstract tuple space.
  - The tuple space is global to the entire system, and to the user, the tuple space looks like a global shared memory. The tuple space represents a shared information space which can be used for coordination.
- **Publish/Subscribe**: This system consists of number of processes connected by a broadcast network.
  - Each process can be producer or consumer of information. When a process has a new information, it publishes the information, ie, it broadcasts the information as a tuple on the network. Each tuple contains the hierarchical subject line containing multiple fields separated by periods. Processes which want to receive information, subscribe to certain subjects.
- **Jini**: This system consists of a large number of self-contained Jini devices, each can offer one or more services.
  - This system has **lookup service and discovery protocol** to find devices. If a new device wants to register, the lookup service responds with a piece of code in JVM. The registration is done for a fixed amount of time - this concept is called *lease*.