Distributed programming

- Distributed systems programming is needed in many of today's applications
  - Web servers, web caches, client-server applications, network devices, …
- From low level to services
  - Sockets (low-level API) TODAY!
  - RPC, RMI (higher-level API discussed earlier)
  - CORBA (also adds services, component model) LATER!
  - EJB, J2EE, (Java version); .net (Microsoft)
  - Jini (Java’s answer to networked services and devices)

Sockets

- Low-level communication interface (API)
- Network Programming
- Defined 1981 Bolt, Beranek and Newman (BBN)
- Originally developed for the C language
- TCP/IP version with Unix called BSD sockets
- Similar API was developed for System V a.k.a TLI
- To some extent similar to file IO

Sockets

- What is a socket?
  - It is a peer-to-peer communication ‘endpoint’ abstraction
  - Hides details of network for programming communication situations
  - Interfaces to some common network protocols (e.g. TCP, UDP, XNS, NetBIOS, IP, etc.)
- Many high level APIs are based on sockets
  - 1985 Sun introduced RPC and NFS (Network File System) over sockets.

Socket types

- Three types:
  - Stream (e.g. interface to TCP),
    - session based reliable service that takes care of connection maintenance, packet reassembly, etc.
  - Datagram (e.g. interface to UDP, NetBIOS),
    - handles independent packets.
    - Unreliable; 5% of datagrams don’t make it.
    - Needs higher level acknowledgement service.
  - Raw (e.g. interface to IP, ICMP, etc.),
    - Programming interface to lower-level protocols

Socket Ports

- On TCP/IP networks:
  - Socket = Internet Address(IP) + Port Address
  - IP is a 32 bit number using dotted string notation, e.g., 206.10.47.09.
  - Port is an "entry point" to an application that resides on a host.
    - Commonly used and defined for server applications

Socket Ports cont.

- Typically a communication is assigned to a port, a process can request/assign to access a port-number.
- TCP and UDP protocols (most common TCP/IP transport protocols) use port numbers 1-1023 that are reserved.
- Standard Internet apps (FTP, TELNET, TFTP, SMTP, SNMP,...) use ports between 1-255.
- > 1023 can be used for new user-level servers
Sockets

- To initiate a connection one must fix the roles: client or server
- A network connection can be connection-oriented or connectionless
  - With a connectionless protocol there is nothing like an “Open” communication; every network IO operation can be with different process/host
  - In a connection based operation a peer-to-peer channel is opened

Connection-oriented use

- Connection is first established between client and server
- System Calls at server side
  - `socket()` // creates endpoint, specify protocol (TCP, UDP, XNS, etc.), specify type (i.e. stream, datagram, etc.) Returns a small integer similar to file descriptor.
  - `bind()` // register and bound network address to socket
  - `listen()` // signal willing to receive connections
  - `accept()` // accept a connection
  - `read()`, `write()` // transfer data

Connection-oriented protocol

- Client side
  - `socket()` // same meaning as in server
  - `connect()` // establish contact between local-remote site
  - `read()`, `write()` // transfer data

Example, connection-oriented client-server sequence

```
socket()                  SERVER
bind()                   CLIENT
listen()                 
accept()                 
read()                   
write()                   

Connection establishment
Data(request)         

read()                   
write()                   
```

System calls connectionless case

- `sendto()` // send datagrams (data packets)
- `recvfrom()` // receive datagrams

- Compared to the connection-oriented `read()` and `write()` these calls have arguments for remote protocol-specific addresses.

Example 2, connectionless client-server sequence of calls

```
socket()                  SERVER
bind()                   CLIENT
recvfrom()               

Data(request)         

sendto()                 
recvfrom()               
```

Comparison

```
Sockets

- To initiate a connection one must fix the roles: client or server
- A network connection can be connection-oriented or connectionless
  - With a connectionless protocol there is nothing like an “Open” communication; every network IO operation can be with different process/host
  - In a connection based operation a peer-to-peer channel is opened

Connection-oriented use

- Connection is first established between client and server
- System Calls at server side
  - `socket()` // creates endpoint, specify protocol (TCP, UDP, XNS, etc.), specify type (i.e. stream, datagram, etc.) Returns a small integer similar to file descriptor.
  - `bind()` // register and bound network address to socket
  - `listen()` // signal willing to receive connections
  - `accept()` // accept a connection
  - `read()`, `write()` // transfer data

Connection-oriented protocol

- Client side
  - `socket()` // same meaning as in server
  - `connect()` // establish contact between local-remote site
  - `read()`, `write()` // transfer data

Example, connection-oriented client-server sequence

```
socket()                  SERVER
bind()                   CLIENT
listen()                 
accept()                 
read()                   
write()                   

Connection establishment
Data(request)         

read()                   
write()                   
```

System calls connectionless case

- `sendto()` // send datagrams (data packets)
- `recvfrom()` // receive datagrams

- Compared to the connection-oriented `read()` and `write()` these calls have arguments for remote protocol-specific addresses.

Example 2, connectionless client-server sequence of calls

```
socket()                  SERVER
bind()                   CLIENT
recvfrom()               

Data(request)         

sendto()                 
recvfrom()               
```
Differences connectionless vs. connection-oriented

- Connection establishing not required
- Blocks until data is received as opposed until connection is established
- Note: Client could also use connect() and read() write() in the connectionless case, but not “real-connect” as it returns immediately, i.e. does not block as there is no connection established.

- Use of the sendto() and recvfrom() as opposed to the read() and write() system calls
- The client registers an address for itself with the bind() to be able to receive messages back (this address is sent with data automatically)

- Connection less is often useful
- Lower overhead, no session creation required
- Discovery type of situations: broadcasting queries to the network and learning who is out there...
- Quick messages (e.g. heartbeat, “I am alive”)
- Can send to 500 nodes, would need 500 sessions with the connection-oriented model
- If one of the messages are not received no problem...

Sockets and Java

- Before RMI and CORBA support has been designed for Java this was the only way to do network programming
- Java.net package contains
  - Set of classes that let you download and manipulate URL objects
  - A native Java implementation of Berkeley (BSD) sockets

- Semantics the same as we discussed earlier
- Java.net contains most of the classes needed
- Java.io contains socket related stream classes
  - A socket may be associated with a “stream” in Java (provided by the Socket class implementation)
  - Streams are fun as they hide devices, its like a data pipe
  - You can use OutputStream, InputStream, and other stream classes derived from these with sockets. Somewhat similar to what you have seen with file access.

Summary Sockets

- Abstraction for network programming.
- Java provides a nice implementation.
- Many known high-level protocols use sockets as underlying communication mechanisms.
- Disadvantages:
  - lots of details to address, many parameters to control.
  - Requires understanding of system issues and network protocols
- Advantage:
  - Best performance in distributed applications, fine-tuning opportunities.

More on the differences

Java sockets

- Programming distributed systems and services
- Objective of this class is to give you an idea of CORBA capabilities and programming approach
**What is CORBA?**

- CORBA = Common Object Request Broker Architecture
- What is CORBA:
  - A “middleware” or intermediate software layer that allow heterogeneous client and server applications to communicate in a distributed system.
  - CORBA works with distributed systems written in: C/C++, COBOL, Java, or ADA.
  - Provides a suite of pre-built complex services
  - An API that allows programmers to build complex distributed services
  - This, as opposed to providing a suite that is a low level network programming API
  - CORBA is similar to Java/IIOP based application servers and Microsoft .net.

**Who stands behind CORBA?**

- It is a product of a consortium - called the Object Management Group (OMG) - including over 700+ companies. http://www.omg.org
  - Notable exception is Microsoft that has its own competing product: DCOM ( and .net & CF - language alternative to Java-announced recently)
  - DCOM = Distributed Component Object Model
  - The Java Enterprise Java Beans (EJB) standard and J2EE application server technology is also a threat to CORBA’s future

**CORBA Orb**

- The ORB forms the “backbone” of CORBA
- It is also called the Object Bus
  - It allows clients to invoke methods on distributed objects and accept return values.
  - There is an ORB on both the client and the server side.
  - Clients and servers can be distributed on a network or located in the same machine.
  - When a client requests a reference for a remote object, the client ORB needs to find the remote object.

**CORBA implementations**

- OMG produces specification not code. Various vendors such as Iona, Oracle, Sun, IBM, Inprise ( Borland), ... have their own CORBA implementations.
- Key components of CORBA:
  - interface definition language or IDL
  - Object Request Broker (ORB)
  - Internet InterORB Protocol (IIOP)
- Additionally ORBs can provide:
  - CORBA services, i.e., naming, life-cycle, events,...
  - CORBA facilities, i.e., system, task, documents management

**CORBA IDL**

- IDL=Interface Definition Language
  - Can be seen as a generic programming language to write standard interfaces for a distributed application
  - it also allows a CORBA “service” or “server” to describe itself in a standard way, independent of actual implementation/OS/language used
  - stub and skeleton(server-side stub) is created based on this interface
  - a client will need to communicate only with this interface.

**Standard benefits ORBs provide**

- Static and dynamic method invocations - it lets you define your method invocations at compile time, or it lets you dynamically discover them at runtime. So you get either strong type checking at compile time or maximum flexibility associated with run-time binding.
  - High-level language bindings - A CORBA ORB lets you invoke methods on server objects using your high-level language of choice. CORBA separates interface from implementation and provides language-neutral data types.
  - Self-describing system - Every CORBA ORB supports an Interface Repository that contains real-time information describing the functions a server provides and their parameters.
Standard benefits of ORBs, 2

- Local/remote transparency - An ORB can run in standalone mode on a laptop or it can be interconnected to every other ORB in the universe using the IIOP protocol services (next slide explains this). The ORB can broker inter-object calls within a single process, multiple processes running in the same machine, or multiple processes running across networks and various OSs.
- Polymorphic messaging - in contrast with RPC, an ORB does not simply invoke a remote function -- it invokes a function on a target object. This means that the effects will be different based on what object receives it. For example, a "setup" method will be different on a business object versus a printer service object.

How do ORBs communicate?

- ORBs communicate through a protocol called Internet InterORB Protocol (IIOP)
  - application-level protocol
  - recall that on top of protocols like TCP/IP we can define application protocols, for example, http is used on the web to communicate between web servers and client browsers.
  - A recent one is iSCSI a protocol to allow SCSI based networked disk drives to run on top of IP.
  - If you come up with a new distributed system you will likely define your own protocol.

Basic CORBA application model

Client

Reference to
CORBA object

Stub

ORB

Server

CORBA object

Skeleton

ORB

IIOP

CORBA services

- Component providers develop their objects without concern for system services, which is provided by the CORBA services
- CORBA services provide a unique approach for build-to-order middleware.
  - Think of services as augmenting the ORBs. Each service is packaged with its own IDL interface.
  - In practice (depending on programming language), when a service is required, then the component providers just subclass the original service classes; service is provided by inheritance
  - For example, a component called "my_account" can be mixed with CORBA services to create a concurrent, persistent, transactional version of "my_account"...

Example of CORBA services

- Life Cycle Service - defines operations for creating, copying, moving components on the object bus
- Persistence Service - provides a single interface for storing components persistently on a variety of storage servers- Object Databases, Relational Databases, and simple files.
- Naming service - allows components on the bus to locate other components by name; also to bound to existing naming contexts, i.e., SUN’s NDS, Novell’s NDS, and the Internet’s LDAP.
- Event Service - allows components on the bus to dynamically register or unregister their interest in specific events.
  - The service defines a well-known object called an event-channel that collects and distributes events among components that know nothing of each other.

Example CORBA services, 2

- Concurrency Control - provides a lock manager that can obtain locks on behalf of either transactions or threads.
- Transaction Service - provides two-phase commit coordination (i.e. when the transaction is distributed the commit must be coordinated among recoverable components)
- Security Service - provides a complete framework for distributed object security. It supports authentication, access control lists, confidentiality, etc.
- Trader Service - provides "Yellow Pages" for objects; it allows objects to publicize their services and bid for jobs.
- Startup Service - enable requests to automatically start up when an ORB is invoked.
Example CORBA services, 3

- **Query Service** - provides query operations for objects. It’s a superset of SQL, the database query language.
- **Time Service** - provides interface to synchronize time in a distributed object environment. It also provides operations for defining and managing time-triggered events.
- **Properties Service** - provides operations that let you associate named values (or properties) with any component. Using this service, you can dynamically associate properties with a component’s state.
- **Licensing Service** - provides operations for metering the use of components to ensure fair compensation for their use.

CORBA facilities

- Are collections of IDL-defined frameworks for applications
- Next step up semantically from the services
- Example
  - data interchange, business object frameworks, internationalization, distributed documents, systems management, information management, ...

Steps 1-4

- 1. Define objects using IDL. This is how objects tell clients what operations are available and how they should be invoked.
- 2. Run IDL file through precompiler
  - e.g., in Java: prompt> idl2java count.idl
  - produces skeletons for the server
- 3. Add the implementation code to the skeletons
- 4. Compile the code - with a CORBA-compliant compiler that is typically capable of generating three types of output files:
  1) import files - describe the objects to an Interface Repository
  2) client stubs - are invoked by client programs that need to remotely access IDL-defined services via ORB
  3) server skeletons - that actually call the methods on the server

Steps 5-7

- 5. Bind the class definitions to the interface repository.
  - Typically, you use a utility to bind.
- 6. Register the runtime objects with the Implementation Repository
  - The Object Adapter records in the Implementation Repository the object reference and type of any object it instantiates on the server. The ORB uses this information to locate active objects or to request the activation of objects in a particular server.
- 7. Instantiate the objects on the server
  - at startup, a server Object Adapter may instantiate server objects dynamically that service remote client method invocations.
  - These runtime objects are instances of the server application classes.
  - CORBA specifies different Object Adapter strategies (outside the scope of this lecture) that are used to create and manage runtime objects.

A simple CORBA app

- Very simple Problem:
  - Count access to server and increment a counter in a server process
  - Count.IDL is the CORBA interface for the server object

```//Count.IDL
module Counter
interface Count
  _getCount: () => long
end interface
```
A simple CORBA app, 2

- Precompile with idl2java count.idl to generate
  - skeletons for the server implementation
  - client side stubs that you need with your client implementations
- A Java interface file corresponding to the IDL file, e.g., Count.java
- A Java class that provides a bind method - used by clients to locate server objects.

A simple CORBA app, 3

- The server-side Count object implementation:
  ```java
  // CountImpl.java
  public class CountImpl implements Count
  {
    // Constructor
    CountImpl() {...}
    // ... (implementation details)
  }
  ```

A simple CORBA app, 4

- The server-side main program implementation
  ```java
  class Server // CountServer.java
  {
    public static void main(String[] args)
    {
      // Start up the ORB
      // ORB orb = ORB.init(args, null);
      ORB orb = ORB.init(args, null);
      // Create the object
      Object object = orb.string_to_object(orb.string_to_idl(string(idl_string)));
      // Create the server object
      ServerImpl server = new ServerImpl();
      // Set up the service in the ORB
      server.set仲(object);
      // Build and service requests
      handle requests();
      // Return to the ORB
      return(0);
    }
  }
  ```

A simple CORBA app, 5

- Example of a client-side code:
  ```java
  public class Client
  {
    public static void main(String[] args)
    {
      // Start up the ORB
      // ORB orb = ORB.init(args, null);
      ORB orb = ORB.init(args, null);
      // Create the server object
      ServerImpl server = new ServerImpl();
      // Get on the service... (details)
      // ... (client side implementation)
      // Get to it in a loop
      int i = -1;
      for (; i < 0; i++)
      {
        // Do something
      }
    }
  }
  ```

Summary

- CORBA - a middleware for heterogeneous distributed computing
- Main Components: IDL, ORB, IIOP
- CORBA provides services, e.g., life-cycle management, events, etc., that can be leveraged in applications.

[Diagram showing Java client connecting to server via IIOP]