Last Class: Introduction

• Distributed Systems
  – A collection of independent computers that appears to its users as a single coherent system

• Hardware concepts
  – Multiprocessors, multi-computers

• Beyond uniprocessor operating systems
  – Distributed OS
  – Network OS
  – Middleware OS
  – Complementary properties

Communication in Distributed Systems

• *Issues in communication (today)*
• Message-oriented Communication
• Remote Procedure Calls
  – Transparency but poor for passing references
• Remote Method Invocation
  – RMIs are essentially RPCs but specific to remote objects
  – System wide references passed as parameters
• Stream-oriented Communication
Communication Between Processes

- **Unstructured** communication
  - Use shared memory or shared data structures

- **Structured** communication
  - Use explicit messages (IPC)

- Distributed Systems: both need low-level communication support *(why?)*

Communication Protocols

- Protocols are agreements/rules on communication
- Protocols could be connection-oriented or connectionless
Layered Protocols

- A typical message as it appears on the network.

Client-Server TCP

(a) Normal operation of TCP.

(b) Transactional TCP.
Middleware Protocols

- Middleware: layer that resides between an OS and an application
  - May implement general-purpose protocols that warrant their own layers
  - Example: distributed commit

Client-Server Communication Model

- Structure: group of servers offering service to clients
- Based on a request/response paradigm
- Techniques:
  - Socket, remote procedure calls (RPC), Remote Method Invocation (RMI)
Issues in Client-Server Communication

- Addressing
- Blocking versus non-blocking
- Buffered versus unbuffered
- Reliable versus unreliable
- Server architecture: concurrent versus sequential
- Scalability

Addressing Issues

- **Question**: how is the server located?
  - Hard-wired address
    - Machine address and process address are known a priori
  - Broadcast-based
    - Server chooses address from a sparse address space
    - Client broadcasts request
    - Can cache response for future
  - Locate address via name server
Blocking versus Non-blocking

• Blocking communication (synchronous)
  – Send blocks until message is actually sent
  – Receive blocks until message is actually received

• Non-blocking communication (asynchronous)
  – Send returns immediately
  – Return does not block either

• Examples:

Buffering Issues

• Unbuffered communication
  – Server must call receive before client can call send

• Buffered communication
  – Client send to a mailbox
  – Server receives from a mailbox
Reliability

- Unreliable channel
  - Need acknowledgements (ACKs)
  - Applications handle ACKs
  - ACKs for both request and reply

- Reliable channel
  - Reply acts as ACK for request
  - Explicit ACK for response

- Reliable communication on unreliable channels
  - Transport protocol handles lost messages

---

Server Architecture

- Sequential
  - Serve one request at a time
  - Can service multiple requests by employing events and asynchronous communication

- Concurrent
  - Server spawns a process or thread to service each request
  - Can also use a pre-spawned pool of threads/processes (apache)

- Thus servers could be
  - Pure-sequential, event-based, thread-based, process-based

- Discussion: which architecture is most efficient?
Scalability

- *Question*: How can you scale the server capacity?
- Buy bigger machine!
- Replicate
- Distribute data and/or algorithms
- Ship code instead of data
- Cache

To *Push* or *Pull*?

- Client-pull architecture
  - Clients pull data from servers (by sending requests)
  - Example: HTTP
  - Pro: stateless servers, failures are each to handle
  - Con: limited scalability
- Server-push architecture
  - Servers push data to client
  - Example: video streaming, stock tickers
  - Pro: more scalable, Con: stateful servers, less resilient to failure
- When/how-often to push or pull?
Group Communication

• One-to-many communication: useful for distributed applications
• Issues:
  – Group characteristics:
    • Static/dynamic, open/closed
  – Group addressing
    • Multicast, broadcast, application-level multicast (unicast)
  – Atomicity
  – Message ordering
  – Scalability

Putting it all together: Email

• User uses mail client to compose a message
• Mail client connects to mail server
• Mail server looks up address to destination mail server
• Mail server sets up a connection and passes the mail to destination mail server
• Destination stores mail in input buffer (user mailbox)
• Recipient checks mail at a later time
Email: Design Considerations

- Structured or unstructured?
- Addressing?
- Blocking/non-blocking?
- Buffered or unbuffered?
- Reliable or unreliable?
- Server architecture
- Scalability
- Push or pull?
- Group communication