DISTRIBUTED COMPUTER SYSTEMS

MESSAGE ORIENTED COMMUNICATIONS

Dr. Jack Lange
Computer Science Department
University of Pittsburgh

Fall 2015

Outline

- Message Oriented Communication
  - Sockets and Socket API
  - Message Passing Interface – MPI
    - Architecture and Design Issues
  - Message Oriented Persistent Communication
    - Message Queues
    - IBM WebSphereMQ
- Stream Oriented Communication
  - Continuous Media Requirements
  - Flow Specification and QoS Support
  - Delay and Jitter Control
  - Packet Loss Control
- Conclusion
**MESSAGE ORIENTED COMMUNICATION**

**SOCKETS**

- A Socket is a communication end point
  - Processes can send and receive messages to and from its socket
- Abstraction over the communication end point that is by the local OS for a specific transport protocol
  - Sending process relies on transport infrastructure which brings message to socket at receiving process

---

**Sockets**

- A Socket is a communication end point
  - Processes can send and receive messages to and from its socket
- Abstraction over the communication end point that is by the local OS for a specific transport protocol
  - Sending process relies on transport infrastructure which brings message to socket at receiving process
Socket Arguments – Protocol

- **Function** `socket(int family, int type, int protocol)`
  - Return > 0 if success and -1 if error

<table>
<thead>
<tr>
<th>Family</th>
<th>Type</th>
<th>Protocol</th>
<th>Actual Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF_INET</td>
<td>SOCK_DGRAM</td>
<td>IPPROTO_UDP</td>
<td>UDP</td>
</tr>
<tr>
<td>AF_INET</td>
<td>SOCK_STREAM</td>
<td>IPPROTO_TCP</td>
<td>TCP</td>
</tr>
<tr>
<td>AF_INET</td>
<td>SOCK_RAW</td>
<td>IPPROTO_ICMP</td>
<td>ICMP</td>
</tr>
<tr>
<td>AF_INET</td>
<td>SOCK_RAW</td>
<td>IPPROTO_RAW</td>
<td>raw IP</td>
</tr>
</tbody>
</table>

Sockets Creation

- The system call socket() is used to create a new socket
  - Int sockfd = socket(PF_INET, SOCK_STREAM, 0);
  - PF_INET = Protocol Family (Internet)
  - SOCK_STREAM = TCP
    - TCP is a reliable protocol
  - SOCK_DGRAM = UDP
    - UDP is a “best effort” protocol, no guaranteed reliability
Overview of the Sockets Interface

Client
- Socket()
- open_clientfd
- connect
- read()
- Close()

Server
- Socket()
- open_listenfd
- Listen()
- Accept()
- write()
- Close()

Connection
- Request
- await connection

Request
- from
- next client

Client Server Session
- Message Oriented Communication
- Message Passing Interface
What is MPI?

- A message-passing library specifications for parallel computers, clusters, and heterogeneous networks
  - Extended message-passing model
- Communication modes
  - Standard, synchronous, buffered, and ready.
- Designed for parallel applications and tailored to **transient** communications
- Provides access to advanced parallel hardware
  - It assumes that serious failures are fatal and do not require automatic recovery
    - For example, process crashes or network disconnect

Group and Context

- MPI assumes communication takes place with a group of processes – (groupID, procID)
  - Use in lieu of a transport-level address
- MPI ties the concepts of **process group** and **communication context** into a **communicator**
MPI Abstractions

- A **process group** is high-level abstraction, visible to users
- A **context** is a system-defined object that uniquely identifies a communicator – Mechanism to isolate messages in distinct libraries and the user programs from one another
  - A message sent in one context can’t be received in other contexts.
  - The communication context is low-level abstraction, not visible to users
- A **communicator** is a data object that specializes the scope of a communication.
  - MPI_COMM_WORLD is an initial communicator, which is predefined and consists of all the processes running when program execution begins

MPI Communication Semantics

- **Point-to-point communication** is the basic concept of MPI standard and fundamental for send and receive operations for typed data with associated message tag.
  - Using the point-to-point communication, messages can be passed to another process with explicit message tag and implicit communication context.
  - Each process can carry out its own code in MIMD style, sequential or multi-threaded.
  - MPI is made **thread-safe** by not using global state.
MPI Communication Primitives

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_bsend</td>
<td>Append outgoing message to a local send buffer</td>
</tr>
<tr>
<td>MPI_send</td>
<td>Send a message and wait until copied to local or remote buffer</td>
</tr>
<tr>
<td>MPI_ssend</td>
<td>Send a message and wait until receipt starts</td>
</tr>
<tr>
<td>MPI_sendrecv</td>
<td>Send a message and wait for reply</td>
</tr>
<tr>
<td>MPI_isend</td>
<td>Pass reference to outgoing message, and continue</td>
</tr>
<tr>
<td>MPI_issend</td>
<td>Pass reference to outgoing message, and wait until receipt starts</td>
</tr>
<tr>
<td>MPI_recv</td>
<td>Receive a message; block if there are none</td>
</tr>
<tr>
<td>MPI_irecv</td>
<td>Check if there is an incoming message, but do not block</td>
</tr>
</tbody>
</table>

MESSAGE ORIENTED COMMUNICATION

MESSAGE QUEUING SYSTEMS
Message-Queuing Model

- MQM, aka Message Oriented Middleware, provides support for **persistent asynchronous** communication
- MOM offers intermediate-term message storage capacity, without requiring either the sender or receiver to be active

MOM Basic Semantics and Interface

- MOM supports **loosely-coupled in time** semantics
  - Senders are given only guarantees that message will eventually be inserted in the receiver’s queue
  - No guarantee about **if** or **when** will messages be read
    - Recipient-behavior dependent
  - Basic interface to a queue in a message-queuing system.

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put</td>
<td>Append a message to a specified queue</td>
</tr>
<tr>
<td>Get</td>
<td>Block until the specified queue is nonempty, and remove the first message</td>
</tr>
<tr>
<td>Poll</td>
<td>Check a specified queue for messages, and remove the first. Never block.</td>
</tr>
<tr>
<td>Notify</td>
<td>Install a handler to be called when a message is put into the specified queue.</td>
</tr>
</tbody>
</table>
MQM General Architecture

- Source and destination queues are distributed across multiple machines
- The relationship between queue-level addressing and network-level addressing.

MQM Generalized Architecture – Routers

- The general organization of a message-queuing system with routers.
Message Brokers

- Common message format, typically used in traditional networks, does not generally work for MQM systems
  - Mostly due to the diversity of applications, which reduces the best common format to a sequence of bytes
  - Brokers – Message broker in a message-queuing system.

MQM Brokers Functionalities

- Message brokers act as an application-level gateway
  - Their main purpose is to convert incoming messages to a format that can be understood by the recipient
  - Properly match end-of-record delimiters between database applications
  - More advanced broker are designed to handle conversations between two different database applications
    - Difficult task, as information and semantic “mapping” between two different applications may not always be possible
- Brokers are commonly used in Enterprise Application Integration for mediation
  - Publish/Subscribe is the typical communication model
  - Brokers “matches” messages to applications’ interests
**WebSphere MQ – MQ Example**

- General organization of IBM's WebSphere MQ message-queuing system – Queue Managers (QM), Message Channels (MC), and Message Channel Agents (MCA)

![Diagram of WebSphere MQ components](image)

**WebSphereMQ MCs and MCAs**

- Message channels are an abstraction of transport level connections.
  - A MC is a unidirectional, reliable connection between a sending and a receiving QM
  - Internet-based MCs are implemented as TCP connections
  - The two ends of a MC are managed by a MCA

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport type</td>
<td>Determines the transport protocol to be used</td>
</tr>
<tr>
<td>FIFO delivery</td>
<td>Indicates that messages are to be delivered in the order they are sent</td>
</tr>
<tr>
<td>Message length</td>
<td>Maximum length of a single message</td>
</tr>
<tr>
<td>Setup retry count</td>
<td>Specifies maximum number of retries to start up the remote MCA</td>
</tr>
<tr>
<td>Delivery retries</td>
<td>Maximum times MCA will try to put received message into queue</td>
</tr>
</tbody>
</table>
WebSphereMQ Message Transfer

- Generalized organization queuing network using routing tables and aliases.
  - Routing Table Entry: <DestMQ, sendQ>, where DestMQ is the name of the destination QM and sendQ is the name of the local send queue

Message Transfer Primitives

- WebSphereMQ Message Queue Interface Primitives

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQopen</td>
<td>Open a (possibly remote) queue</td>
</tr>
<tr>
<td>MQclose</td>
<td>Close a queue</td>
</tr>
<tr>
<td>MQput</td>
<td>Put a message into an opened queue</td>
</tr>
<tr>
<td>MQget</td>
<td>Get a message from a (local) queue</td>
</tr>
</tbody>
</table>
Stream Communication Model

- Support for data streams, time-dependent information such as audio and video, requires communication models that go beyond those that support independent and complete units of information.

General Architecture for Streaming Stored Media over a Network
Data Streams – Point-to-Point

- Setting up a stream directly between two devices.

Data Streams -- Multicasting

- An example of multicasting a stream to several receivers.

[Diagram of point-to-point data streams]

[Diagram of multicast data streams]

(b)
Streams, flows and QoS

- The Quality of Service (QoS) of an application is the “look and feel” of how the stream will flow

- QoS Support depends on two components
  - Specification of the flow/stream
  - Enforcing the QoS requirements specification of the supported flows

- The QoS support implementation depends on many, many factors,
  - It may require the “Flow Admission Protocol”
    - Given a new flow specifications, and a set of currently supported flows, a decision has to be made whether the new flow is to be accepted or rejected

Streams and Quality of Service

- QoS Flow Specification Parameters
  - The required bit rate at which data should be transported.
  - The maximum delay until a session has been set up
  - The maximum end-to-end delay.
  - The maximum delay variance, or jitter.
  - The maximum round-trip delay.
QoS Specification and Services

- A flow specification model and services

<table>
<thead>
<tr>
<th>Characteristics of the Input</th>
<th>Service Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum data unit size (bytes)</td>
<td>Loss sensitivity (bytes)</td>
</tr>
<tr>
<td>Token bucket rate (bytes/sec)</td>
<td>Loss interval (µsec)</td>
</tr>
<tr>
<td>Token bucket size (bytes)</td>
<td>Burst loss sensitivity (data units)</td>
</tr>
<tr>
<td>Maximum transmission rate (bytes/sec)</td>
<td>Minimum delay noticed (µsec)</td>
</tr>
</tbody>
</table>

Specifying QoS – Token Bucket

- The principle of a token bucket algorithm.

![Diagram of token bucket algorithm](image)
Enforcing QoS – Binding Jitter

- Buffers can be used to reduce jitter
  - Jitter represents the delay variation between consecutive packets
  - Typically measured as the variance of packet interarrival times

```
Packet departs source: 1 2 3 4 5 6 7 8
Packet arrives at buffer: 1 2 3 4 5 6 7 8
Packet removed from buffer: 1 2 3 4 5 6 7 8
```

Enforcing QoS – Mitigating Packet Loss

- The effect of packet loss (a) non interleaved transmission and (b) interleaved transmission.

```
(a) Lost packet
Sent: 1 2 3 4 5 6 7 8 13 10 11 12
Delivered: 1 2 3 4 5 6 7 8 9 10 11 12
Gap of lost frames:

(b) Lost frames
Sent: 1 5 9 13 2 6 10 14 3 7 11 15
Delivered: 1 2 3 4 5 6 7 8 9 10 11 12
Lost frames:
```
Setting Up a Stream

- Resource reservation in a distributed system
- OS must support mechanism
- Policy can be implemented at the user level

Reservations
- Statistical (good on average) vs guaranteed (good for important stuff)
- No standards, much research, no prevalent results

Synchronization Mechanisms

- Synchronization of data and of flows is difficult, mainly due to the unpredictability of computing systems
  - Interrupts may disturb media continuity
  - Priority inversion may occur – High priority task blocked by a low priority task
  - Different flows must be synchronized

- Application-level Synchronization
  - Hardware can be used to help specially when synchronization requirements are strict
    - Separate streams should be played within 30 μ-seconds
Synchronization Mechanisms

- Synchronization supported by high-level interfaces.
- Middleware multiplexes all substreams into a single stream and demultiplex the streams at the receiver.
- Synchronization is handled at multiplexing/demultiplexing point
- Example: MPEG

Conclusion

- Message Oriented Communication
  - Sockets and Socket API
  - Message Passing Interface – MPI
    - Architecture and Design Issues
  - Message Oriented Persistent Communication
    - Message Queues
    - IBM WebSphereMQ
- Stream Oriented Communication
  - Continuous Media Requirements
  - Flow Specification and QoS Support
  - Delay and Jitter Control and Reliability