TCP/IP Networking: Socket I/O

17 Mar 2010
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY
WESTERN
UNIVERSITY

TCP client-server

- TCP is connection-based:
  - Phone analogy
  - Initial setup, but subsequent packets do not need to specify destination again
  - Server: waits, listens for client
  - Client: initiates connection (phone call)
  - Once connection is established, communication may be two-way (send/receive)
  - Either client or server may terminate

Making a TCP Server in Java

- java.net.ServerSocket object
    - **server = new ServerSocket(*port*, *maxcl*);**
  - *maxcl:* queue length (reject extra clients)
  - BindException raised if port invalid or in use
- Bind socket (start listening) (blocking):
    - **connection = server.accept();**
  - Returns a java.net.Socket object
- Communicate via streams:
    - **connection.getInputStream();**
    - **connection.getOutputStream();**

Communicating with streams

- Both client and server may send or receive:
  - **conn.getInputStream()**
  - **conn.getOutputStream()**
- Communicate via text streams:
  - **new Scanner(conn.getInputStream());**
  - **new PrintWriter(conn.getOutputStream());**
- Or object streams:
  - **new ObjectInputStream(**
    conn.getInputStream());
  - **new ObjectOutputStream(**
    conn.getOutputStream());

How do we accept clients?

- Iterating server: only one client at a time
  - One operator answering phones
  - Simplest to implement
- Forking server:
  - Split off a child thread for each connection
  - Original master thread continues to listen
  - Switchboard
- Concurrent single server:
  - Use select to simultaneously wait on all open socket IDs

More on forking server

- Multiple threads running concurrently
- Master thread listens on port
- When a client connects, fork off a thread
  - Thread handles communication with that client
- Master thread continues listening for other connections (switchboard)

- Overhead in forking new threads: so keep pool of available threads, and reuse dormant threads

Connectionless client/server

- TCP is connection-oriented
- UDP is connectionless
  - Send data one packet at a time
    - Similar to envelopes through CanadaPost
    - Fragment larger data into multiple packets
  - Packets might:
    - Not arrive at all
    - Arrive out of order
    - Get duplicated
  - Less overhead, better latency and possibly better throughput

Receiving a UDP packet

- Create a DatagramSocket (in java.net):
  - **sock = new DatagramSocket(*port*);**
- Create a DatagramPacket to store the data:
  - **byte payload[] = new byte[100];**
  - **packet = new DatagramPacket(payload, payload.length);**
- Wait (block) for a packet:
  - **sock.receive(packet);**
- Read info from packet:
  - **packet.getData(), .getLength(), .getAddress(), .getPort()**

Sending a UDP packet

- Prepare payload:
  - **String msg = “Hello, World!”;**
  - **byte[] payload = msg.getBytes();**
- Package payload:
  - **packet = new DatagramPacket(payload, payload.length, *hostname*, *port*);**
- Send packet:
  - **socket.send(packet);**