CS 3204
Operating Systems

Lecture 41
Godmar Back

Announcements
• Project 4 due Wed, May 3, 11:59pm
• Read chapter 13 (networking)
• Final Exam Fri, May 5, 1:05pm-3:05pm
• Wednesday, May 3:
  – Final review session (Q&A only)
  – Course Evaluations

Networking
(Most slides from Kurose/Ross: Computer Networking – A Top Approach Featuring The Internet)

Demultiplexing
• End systems must decide which layer instances should process an incoming packet.
• Layer k has “type information” in header to say which instance of layer k+1 to pick.
• Issues: speed and flexibility

Typical Implementation

Socket Programming
UDP & TCP
Network Socket Programming

**Socket**: (narrow definition:) a door between application process and end-end-transport protocol (UDP or TCP)

- Host or server
- Stack with buffers, variables socket controlled by application developer
- Stack with buffers, variables socket controlled by operating system
- Internet

Socket Programming

**Socket API**
- Introduced in BSD 4.1 UNIX, 1981
- Explicitly created, used, released by apps
- Used for both local and remote communication

**BSD Socket API**
- API – Application Programming Interface
  - Provides access to services
  - Specified in C language
  - Implemented on many platforms in one way or the other
    - (Windows: WinSock2, CSocket MFC classes for BSD-like look)
- Sockets (in Unix) are file descriptors
  - General idea: writing to the socket is sending data to network, reading from socket is receiving data
  - Good because read(2), write(2), close(2) and others (select(2), poll(2), ioctl(2), SIGIO, fcntl(2)) can be reused
  - Bad because suggest orthogonality if where there is none
- Other languages provide separate mapping, often thin veneers over BSD sockets (e.g., java.net.Socket)

**UDP**
- Service provided
  - Demultiplexing
  - Payload checksum
- Passes segments straight to IP
  - No congestion control
- So simple it fits on one slide

**Addressing**
- For UDP/IP or TCP/IP socket communication, generally need 4 parameters:
  - Source Identifier (32-bit IP Address)
  - Source Port (16-bit)
  - Destination Identifier (32-bit IP Address)
  - Destination Port (16-bit)
- Notice that the relationship of “local” and “remote” (also called “peer”) to source/destination depends on direction of communication
- Note:
  - UDP uses only Destination (IP+Port) for demultiplexing
  - TCP uses Source + Destination
    - (quadruple: Src IP, Src Port, Dist IP, Dist Port)

**UDP Sockets: Overview**
UDP Demultiplexing

TCP Sockets
- Provide reliable byte-stream abstraction
  - In-order, reliable delivery of bytes
- Connection-oriented
  - Client must connect(2)
  - Server performs "passive open" using accept(2)

TCP Sockets: Overview

Server Models
- Should more than one more client at a time be supported?
  - No: use iterative approach: one at a time
- If Yes: How do we manage \( n \) clients (and be able to accept more at the same time)?
  - Option 1: use multiple execution contexts (aka "thread-based", concurrent model)
  - Option 2: multiplex multiple connections in single execution context (aka "event-based", achieves "apparent concurrency")

Multiple Contexts
- Option A: fork a new process for every connection on-demand
- Option B: fork a new thread for every connection to handle it
- Option C/D: pre-fork a certain number of processes (or threads) and hand connections to them
Handling multiple clients using multiple execution contexts

Q.: When would you use which?

A/B: # grows & shrinks

C/D: fixed #

Multiplexing Multiple Connections

• Problem: need to avoid blocking
• Different solutions:
  – Always test before you read/write (would I block?)
    • NB: does not require use of nonblocking mode
  – Use socket in nonblocking mode: try to read, let it fail
    if it would block
    • Then try again later (when?), or:
      use in combination with notification: send a signal or event if
      the socket becomes readable.
• Many combinations possible