Networking Lecture

Today's Topics

- Computer Networking
- Internet Protocols
- IP Sockets Programming
- NetBios Programming (Obsolete, Broken on Windows 2000)
- Advanced Networks

Computer Networks

- **Local-Area and Wide-Area Networks (LANs/WANs)**
  Ethernet: 10/100/1000 Mbps (Local-Area)
    - Packets broadcast on shared media (the 'Ether')
    - Coax (10Base-2): Terminators prevent signal reflection
    - Twisted Pair (10/100/1000Base-T): Hubs or switches retransmit packets to links
    - No Quality of Service (QoS) Guarentees
- **Data Delivery**
    - Broadcast: One-to-all
    - Multicast: One-to-many (using group name)
    - Unicast: One-to-one (each host has unique name)
- **Data Transport Protocols**
    - Datagram
        - Packets may be lost
        - Packets may be reordered (except ATM)
        - Can Broadcast, Multicast, Unicast
        - Example: UDP
    - Connection-Oriented
        - Reliable connection between endpoints
        - Lost packets are retransmitted
        - Limited to point-to-point (Unicast) links
        - Example: TCP
- **Programming Models**
    - Client/Server (typical case)
        - Server is a centralized, permanent resource
        - Clients are short-lived, stateless programs
        - Unicast Connections between server and client
    - Peer-to-Peer

        - All Programs share information
        - Multicast or broadcast data between hosts

- **Network Adapter & Software**

    - Hardware address filters
    - Interrupt On Incoming Packet
    - Demultiplex incoming packet
    - Deliver data to appropriate application / higher level protocol (eg IP)

Internet Protocols

- **Internet Protocol (IP) is a global standard**

    - Allows for a common address space across entire Internet
    - 32-bit address, but not every address is usable
    - Packetized: data is split into segments for transmission
    - Unreliable transport, so packets may be:
        - lost
        - duplicated
        - arrive out of order
    - Independent of underlying hardware (may run on ethernet, ATM, etc)
    - We don't use IP directly, we mostly use two protocols built on it

- **UDP (UDP/IP) -** ***User Datagram Protocol***

    - **Still unreliable**, but allows for multiplexing
        - Multiplexing = sharing of a single IP address by multiple programs
        - Performed by using a **Port number**, a 16 bit unsigned integer
        - Can have different source and destination ports
        - The operating system demultiplexes each incoming packet:
            - looks at the port number
            - sends the packet to the program listening on that port
    - Unidirectional, application needs to establish reverse path
    - Checksummed to prevent data corruption in packet
        - But is still unreliable (packets may be lost), just like pure IP
        - We'd like a reliable protocol for sending large amounts of data

- **TCP (TCP/IP) -** ***Transmission Control Protocol***

    - A **reliable** stream transport protocol
        - Streamed connection: don't really see individual packets
        - Packets are guaranteed to arrive in the order sent, with no losses
        - Provides a "virtual circuit" connection
        - Full duplex once connection is established
        - Disadvantage: variable delay (sometimes *very* variable)
    - Has multiplexing, just like UDP (16-bit port numbers)
    - For how this actually works internally, take ECE 338!

IP "Sockets" Programming

- **So how do we actually *use* TCP/IP and UDP/IP?**
  - Most used: Sockets API
    - Introduced in 1981 in BSD 4.1 (Unix clone)
    - Implemented as system library calls
    - TCP and UDP interfaces very similar
    - Portable Microsoft implementation: WinSock

- **Byte Order**
  - As you all know, x86 machines are little endian.
  - IP protocols use network order, which is big endian.
  - Use functions to convert between "host" (little endian) byte order and network byte order:
    - Socket_htonl, Socket_htons - convert 32-bit long, 16-bit short from host to network order
    - Socket_ntohl, Socket_ntohs - vice-versa
  - **All socket functions require parameters (particularly the address and port) in network order**

- **SOCKADDR structure**
  - Contains both the port (SOCKADDR.Port) and the IP address (SOCKADDR.Address)
  - Many sockets functions take a pointer to this structure instead of two seperate values

- **Address Conversion**
  - Need a way to convert to and from human-readable (ASCII) names to IP addresses.
  - Some functions:
    - Socket_inet_addr: convert a dotted decimal ("127.0.0.1") into an 32-bit address
    - Socket_inet_ntoa: convert an 32-bit address into dotted decimal ASCII
    - Socket_gethostbyname: converts an english host name ("www.uiuc.edu") into an IP address
    - Socket_gethostbyaddr: converts an IP address into an english hostname
    - Socket_gethostname: gets the local host's name (call gethostbyname to get the local IP)

- **Socket Creation and Setup**
  - Socket_create: creates a socket of a particular type (called socket() in standard implementations)
  - Waiting for connection (server) side:
    - Socket_bind: binds a socket to a local IP address and port number
    - Socket_listen: puts socket into a passive state (waiting for a connection)
    - Socket_accept: accepts a new connection (TCP only)
  - Connecting (client) side:
    - Socket_connect: connect to another (remote) socket (TCP only)

- **Sending and Receiving Data**
  - TCP way (after connection is established)
    - TCP is streaming, so we can view the connection as just a stream of bytes
    - Socket_recv: receive data
    - Socket_send: send data
  - UDP way
    - Each transmission is a new packet (unreliable, so it may be lost)
    - Socket_recvfrom: gets a single incoming packet
    - Socket_sendto: sends a single packet to another (remote) socket

- **Closing a connection**
  - Socket_close: closes a socket completely (works for both TCP and UDP)
  - Socket_shutdown: closes one direction of the connection (TCP only)

- **Examples**
  - Sockets programming (and network programming in general) is difficult.
  - Examples are *very* helpful!

    - Because sockets are a standard interface, many examples are available on the web (basically all in C):
        - WinSock application samples
        - Tutorial: Datagrams
        - Tutorial: Connections
        - CIS 307: Sockets
    - PModeLib also includes a few example programs in assembly (in the V:\ece291\pmodelib\examples directory)
        - tcpweb: retrieves the ECE 291 webpage and prints it to the screen
        - tcpsrv: a simple TCP server
        - tcpcli: a simple TCP client (connects to tcpsrv)
        - udpsrv: a simple UDP server
        - udpcli: a simple UDP client (connects to udpsrv)

- **Tips**
    - Pick your protocol (TCP or UDP) carefully!
        - TCP tends to be best for:
            - transaction-based communications
            - sending a large stream of data that must be received perfectly (eg a file)
        - UDP tends to be best for:
            - Anything realtime (game updates, sound files, etc)
    - Structure data to transmit over the network: the smaller the better
    - Use callback function (interrupt-driven network I/O) to not stop your main loop
        - PModeLib Socket_SetCallback and Socket_AddCallback allow you to do this in PModeLib.
        - You can trigger on different sockets for different events, but *all* events get handled by *one* callback function.

- **References**
    - As sockets is a standard interface to using TCP/IP and UDP/IP, there are some excellent references:
        - Windows Sockets 1.1 Reference
        - CS/ECE 338 Class Notes
        - PModeLib documentation
    - NOTES:
        - PModeLib Sockets functions are based on WinSock
        - Note: some PModeLib versions of WinSock functions have been simplified to make them easier to use in assembly.

NetBIOS (Obsolete, Broken on Windows 2000)

- **Local Area Network *service***

    - Independent of underlying hardware
        - Ethernet
        - Token Ring
        - FDDI
    - Independent of underlying Protocols
        - Standalone packets
        - Encapsulation in IP
    - Designed for Local Area Network
        - Assumes single local network segment
    - Implementation
        - DOS Packet Driver
        - Serice provided by WinNT, Win95, WFWG

- **NetBIOS Provides**
    - Naming Service
        - Provides Unique names on Network
        - Global Namespace
    - Multiple types of communication
        - Datagram Broadcast
        - Datagram Multicast
        - Datagram Unicast
        - Reliable Connection
- **Communication with NetBIOS**
    - Interrupt 5C
- **Network Control Block (NCB)**
    - Basic NetBIOS Data Structure
    - Command Field: Determines what action to take
    - Inputs/Outputs/Pointers: All passed through NCB
- **NetBIOS Response Methods**
    - Blocking Functions: Wait until even occurs
    - CallBack Functions: Register function to call when event occurs
      Operates just like an interrupt
- **NetBIOS Names**
    - 16-character NetBIOS identifiers
    - Local Names: Unique across the LAN (enforced by NetBIOS)
    - Group Name: Shared among multiple machines (Multicast)
- **Examples**
    - **Broadcast/Multicast Datagrams**
        - Sender & Receiver ADD NAME
        - *Loop while running*
            - Sender SENDS Broadcast Datagram
            - Receiver RECEIVES Datagram
        - Sender & Receiver REMOVE NAME
    - **Unicast Connection**
        - Sender & Receiver ADD NAME
        - Receiver LISTENS
        - Sender CALLS *(Connection Established)*
        - *Loop while running*
            - Sender SENDS
            - Receiver RECEIVES
        - Sender or Receiver HANGUP *(Connection Terminated)*
        - Sender & Receiver REMOVE NAME
- **Complete Documentation:** [CBIS NetBios Programmers Reference]

- **Programming with NetBIOS (Doesn't work under Windows 2000)**
  - **Lockwood's NetLIB:** *(API to make your life easy!)*
    - NetLIB readme file
    - NetLIB: ASM source code & examples
  - **Important Variables**
    - **grp_name**: Multicast Group name *(Default=ECE291NetLib$$$$)*
    - **my_name**: My unique network id *(Default=ECE291Player0$$$)*
    - **TXBUffer**: Transmission buffer
      Load with data that you want to transmit then call *SendPacket*.
    - **RXBUffer**: Receive Buffer
      Filled with data from incoming network packet network just before your *netpost* function is called.
  - **Procedures**
    - **NetINIT**: Call at start of your program
    - **netpost**: Callback function called whenever a datagram arrives.
      - Called with
        - BX = pointer to receive buffer
        - AX = length of data
      - Because this routine is called from an interrupt, it must:
        - Preserve all registers
        - Avoid DOS and LIB291 calls
    - **SendPacket**: Call with AX = Length of TXBuffer to transmit data
    - **NetRelease**: Call at the end of your program
  - **PModeLib also has similiar functions**

Advanced Networks

- **ATM: Asynchronous Transfer Mode** (Local & Wide-Area)
  - Data transmitted as 53-byte cells
  - Messages transmitted using Adapatation Layer (AAL)
  - Transmits data at rate of 155 Mbps (OC3), 622 Mbps (OC12), 2.4 Gbps (OC48), or 10 Gbps (OC192) *per host* and shared on virtual circuits.
  - Preserves *Quality of Service*, allowing integrated data (voice, video, IP) to carried over common network.
  - See the iPOINT Testbed for information about ATM research on this campus.

Return to ECE291 Lecture Index
