Internet Programming & Protocols
Lecture 13

Network programming in Java, Perl, Windows

Review

assignment 4 and 5 and 6

www.cs.utk.edu/~dunigan/ipp/

Java network programing

- interpreted language
- applets or jre
- applets restricted to communicating to server only
- RPC and more with RMI
- more network features in java 1.2, 1.3, 1.4 …
- object-oriented interface

TCP echo client

```
/* invoke with target hostname */
import java.io.*;
import java.net.*;

public class EchoClient {
  public static void main(String[] args) throws Exception {

    Socket echoSocket = new Socket(args[0], 6789);
    PrintWriter out = new PrintWriter(echoSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(new InputStreamReader(
      echoSocket.getInputStream()));
    BufferedReader stdIn = new BufferedReader(
      new InputStreamReader(System.in));
    String userInput;

    while ((userInput = stdIn.readLine()) != null) {
          out.println(userInput);
          System.out.println("echo: " + in.readLine());
    }
    echoSocket.close();
  }
}
```

TCP echo server

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception {
    String clientSentence;
    String capitalizedSentence;

    ServerSocket welcomeSocket = new ServerSocket(6789);

    while(true) {

          Socket connectionSocket = welcomeSocket.accept();

          BufferedReader inFromClient =
           new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
          DataOutputStream  outToClient =
            new DataOutputStream(connectionSocket.getOutputStream());
          clientSentence = inFromClient.readLine();
          capitalizedSentence = clientSentence.toUpperCase() + '\n';
          outToClient.writeBytes(capitalizedSentence);
          connectionSocket.close();
    }
  }
}
```

UDP client

```
import java.io.*;
import java.net.*;
class UDPClient {
  public static void main(String args[]) throws Exception {
    BufferedReader inFromUser =
    new BufferedReader(new InputStreamReader(System.in));
    DatagramSocket clientSocket = new DatagramSocket();

    InetAddress IPAddress = InetAddress.getByName(args[0]);
    byte[] sendData = new byte[1024];
    byte[] receiveData = new byte[1024];
    String sentence = inFromUser.readLine();
    sendData = sentence.getBytes();

    DatagramPacket sendPacket =
      new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
    clientSocket.send(sendPacket);
    DatagramPacket receivePacket =
      new DatagramPacket(receiveData, receiveData.length);
    clientSocket.receive(receivePacket);
    String modifiedSentence = new String(receivePacket.getData());
    System.out.println("FROM SERVER:" + modifiedSentence);
    clientSocket.close();
  }
}
```

UDP server

```
import java.io.*;
import java.net.*;

class UDPServer {
public static void main(String args[]) throws Exception {
    DatagramSocket serverSocket = new DatagramSocket(9876);
    byte[] receiveData = new byte[1024];
    byte[] sendData  = new byte[1024];

    while(true) {
          DatagramPacket receivePacket =
            new DatagramPacket(receiveData, receiveData.length);
          serverSocket.receive(receivePacket);
          String sentence = new String(receivePacket.getData());
          InetAddress IPAddress = receivePacket.getAddress();
          int port = receivePacket.getPort();

          String capitalizedSentence = sentence.toUpperCase();

          sendData = capitalizedSentence.getBytes();
          DatagramPacket sendPacket =
            new DatagramPacket(sendData, sendData.length, IPAddress, port);
          serverSocket.send(sendPacket);
    }
  }
}
```

Socket options

various Socket methods for socket options

- setTcpNoDelay(boolean on)
- setSoLinger(boolean on, int linger)
- setSendBufferSize(int size)
- setReceiveBufferSize(int size)
- Other options supported: **SO_REUSEADDR SO_KEEPALIVE**
  - **getOption**(int optID)
  - **setOption**(int optID, Object value)

```
SocketImpl s;
 ...
s.setOption(SO_LINGER, new Integer(10));
```

RMI -- remote objects/methods (RPC)

Perl socket programming

- interpreted language
- multi OS
- CGI scripts and such
- conventional BSD socket
- perl5.004, object-oriented sockets

TCP client

```
#!/usr/local/bin/perl5.004
use Socket;
my ($remote,$port, $iaddr, $paddr, $proto, $line);

$remote  = shift || 'localhost';
$port    = shift || 2345;  # random port
if ($port =~ /\D/) { port = getservbyname($port, 'tcp') }
     die "No port" unless $port;
$iaddr   = inet_aton($remote)               || die "no host: $remote";
$paddr   = sockaddr_in($port, $iaddr);

$proto   = getprotobyname('tcp');
socket(SOCK, PF_INET, SOCK_STREAM, $proto)  || die "socket: $!";
connect(SOCK, $paddr)    || die "connect: $!";
while (defined($line = <SOCK>)) {
    print $line;
}

close (SOCK)            || die "close: $!";
exit;
```

TCP server

```
#!/usr/local/bin/perl5.004
use strict;
use Socket;
use Carp;

sub logmsg { print “$0 $$: @_ at ", scalar localtime, "\n" }

my $port = shift || 2345;
my $proto = getprotobyname('tcp');
$port = $1 if $port =~ /(\d+)/; # untaint port number

socket(Server, PF_INET, SOCK_STREAM, $proto)        || die "socket: $!";
setsockopt(Server, SOL_SOCKET, SO_REUSEADDR,pack("l", 1))   || die "setsockopt: $!";
bind(Server, sockaddr_in($port, INADDR_ANY))        || die "bind: $!";
listen(Server,SOMAXCONN)                            || die "listen: $!";

logmsg "server started on port $port";
my $paddr;
for ( ; $paddr = accept(Client,Server); close Client) {
    my($port,$iaddr) = sockaddr_in($paddr);
    my $name = gethostbyaddr($iaddr,AF_INET);

    logmsg "connection from $name [", net_ntoa($iaddr), "] at port $port";

    print Client "Hello there, $name, it's now ", scalar localtime, "\n";
}
```

Newer IO module

```
use IO::Socket;
$remote = IO::Socket::INET->new(
    Proto    => "tcp",
    PeerAddr => "localhost",
    PeerPort => "daytime(13)",
) or die "cannot connect to daytime port at localhost";
while ( <$remote> ) { print }
```

Windows sockets

- console C (blocking)
- event driven C
- C++ (MFC classes)

Console C

- For UNIX die hards, applications runs under command window (DOS)
- borland or visual C++
- nmake from command window
- run from command window
- Also cygwin (UNIX like environment with C compilers) or Microsoft's "UNIX services" package

---

Generic make

- for single source module

```
CC=cl
CFLAGS=-c -DSTRICT -DWIN32 -D_CONSOLE -O2  -Zp

LINKER=link
GUIFLAGS=-SUBSYSTEM:console

LIBS=kernel32.lib wsock32.lib
(APP).exe : (APP).obj
(LINKER) (GUIFLAGS) -OUT:(APP).exe (APP).obj (LIBS)



plus i have a little bld.bat  (just say   bld bob  to compile bob.c)

@ echo off
set app=%1
nmake generic.mak
```

Modifying UNIX C network program for Windows

- don't need all those includes,
- just windows.h and winsock.h
- initialize WinSock DLL and cleanup
- when done WSAStartup()/WSACleanup()
- various winsock DLL's provide different services/implementations
- WSAGetLastError() to get "errno"
- replace read/write with recv/send
- Get rid of fork() gettimeofday() ….
- Window's "UNIX services" package can provide a compatibility base

---

TCP server

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include<winsock.h>
main(argc,argv)
int argc;
char *argv[];
{
    sockinit();
    tcprecv();
    WSACleanup();
}
```

```
sockinit()
{
WORD wVersionRequested;
WSADATA wsaData;
int err;

   wVersionRequested = MAKEWORD( 2, 0 );

   err = WSAStartup( wVersionRequested, &wsaData );
   if ( err != 0 ) {
  /* Tell the user that we couldn't find a usable */
  /* WinSock DLL.                                  */
      printf("startup failed %d\n",err);
      return;
   }
}
```

---

tcprecv()

```
tcprecv()
{
int lth;
SOCKET lsd, sd;
SOCKADDR_IN in,fr;

    lsd =socket(AF_INET,SOCK_STREAM,0);
    in.sin_family = PF_INET;
    in.sin_port = htons(port);
    in.sin_addr.s_addr = htonl(INADDR_ANY);
    bind(lsd,&in,sizeof(in));
    lth = sizeof(fr);
    listen(s,5);
    sd = accept(lsd, &fr, &lth);
    while(lth > 0){
          lth=recv(sd,buff,MAXL,0);
          send(sd,buff,lth,0);
    }
    shutdown(lsd,2);
    closesocket(lsd);
    shutdown(sd,2);
    closesocket(sd);
}
```

TCP client

```
...
s =socket(AF_INET,SOCK_STREAM,0);
in.sin_family = PF_INET;
in.sin_port = htons(port);
in.sin_addr.s_addr = GetAddr(dst);
err=connect(s,(LPSOCKADDR)&in,sizeof(in));
if (err<0)
{
    printf("connect err %d host %s\n",WSAGetLastError(),dst);
    return;
}
...
send(s,buff,rlth,0);
rlth = mread(s,buff,lth);  /* recv til lth*/
...
shutdown(s,2);
closesocket(s);
```

GetAddr()

```
GetAddr (LPSTR szHost) {
LPHOSTENT lpstHost;
u_long lAddr = INADDR_ANY;

/* check that we have a string */
if (*szHost) {

    /* check for a dotted-IP address string */
    lAddr = inet_addr (szHost);

    /* If not an address, then try to resolve it as a hostname */
    if ((lAddr == INADDR_NONE) &&(strcmp (szHost, "255.255.255.255"))) {

          lpstHost = gethostbyname(szHost);
          if (lpstHost) {  /* success */
                    lAddr = *((u_long FAR *) (lpstHost->h_addr));
          } else {
                    lAddr = INADDR_ANY;   /* failure */
          }
    }
}
return (lAddr);
} /* end GetAddr() */
```

Porting from UNIX

may vary by winsock version

- only need windows.h
- socket are NOT file handles, use send/recv not write/read
- no fork(), use threads:
  - _beginthread() and link with -MT
- no syslog(), or errno/perror
- replace bcopy() etc. with memcpy() etc.
- select() can't be used as timer
- worry about byte-order -- ntohl/htonl
- no UNIX domain (winsock 1.1)
- no RPC (not included)
- no signal()
- event driven I/O WSAxxxx()
- most setsockopt() OK

Windows supports multicast and RAW/ICMP sockets

Winsock 2

wsock32.dll

- multi protocol support (ATM)
- scatter/gather
- quality of service
- overlapped I/O (more WSAcalls())
- probably need other packages for RAW sockets or data-link layer support (windump)

Winsock layers

```
+---------------+  +-----------------++-----------------+
|   WinSock 2   |  | 16-bit WinSock  || 32-bit WinSock  |
|  Application  |  | 1.1 Application || 1.1 Application |
+---------------+  +-----------------++-----------------+
|                 |                  |
|         +-----------------++-----------------+ <-- WinSock
|         |   WINSOCK.DLL   ||  WSSOCK32.DLL   |     1.1 API
|         |    (16-bit)     ||   (32-bit)      |
|         +-----------------++-----------------+
|                                  |
+-------------------------------------------------------+ <-- WinSock
|                  WS2-32.DLL (32-bit)                  |     2.0 API
+-------------------------------------------------------+
|
+-------------------------------------------------------+ <--- WinSock
|+-----------++--------------++------------++----------+|      2.0 SPI
||  TCP/IP   || TCP/IP-based || additional ||Layered SP||     /
|| transport ||   namespace  ||   service  |+----------+|    /
||  service  || service pro- ||  providers |+----------+| <--
||  provider || vider (DNS)  || (SPX, etc.)|| Any TSP  ||
|+-----------++--------------++------------++----------+|
```

Measuring elapsed time

GetTickCount() returns milliseconds

can get microseconds from performance counter

```
double
seconds()
{
    static unsigned int mhz=0;
    LARGE_INTEGER t;
    double s;

    if (mhz ==0 ){
          QueryPerformanceFrequency(&t);
          mhz = t.LowPart;
    }
    QueryPerformanceCounter(&t);
    s = t.QuadPart;
    return(s/mhz);
}
```

Asynch I/O

non-console -- full Windows GUI

- windows are objects that communicate with messages
- asynch sockets uses Windows messages
- WSAAsynchGetHostByName(), WSAAsynSelect()
- handle with Windows event handlers
- GUI friendly
- typically slower
- then there are the C++ socket classes,
  - CSocket:: and CAsyncScoket::

variations

- Other languages
  - Visual basic
  - Python
  - php
- Remote procedure calls
  - Hides details of socket stuff
  - Usually simple requuest/reply (UDP)
  - Handles data conversion (XDR)
  - Service registry done through portmap
  - Java RMI

review

- Lectures
- required reading
- Concepts
- Tools
- Slow us down

Plan of attack

- Network overview ✓
- BSD sockets and UDP ✓
- TCP ✓
  - Socket programming
  - Reliable streams
  - Header and states
  - Flow control and bandwidth-delay
  - Measuring performance
  - Historical evolution (Tahoe …SACK)
  - Congestion control
- Network simulation (ns)
- TCP accelerants
- TCP implementations
- TCP over wireless, satellite, …

LECTURES

1 overview, class mechanics, networks 101
2 Ethernet, IP, ARP
3 IP routing, tcpdump/ethereal ICMP ping/traceroute
4 UDP, BSD sockets, client/servers
5 UDP, DNS
6 TCP socket programming
7 reliable streams, TCP header
8 TCP states, flow control, bandwidth-delay
9 performance tools
10 nagle, delayed ACKs, timers, RTT estimation, TCP slow-start
11 TCP congestion control, TCP Tahoe
12 TCP Reno, NewReno, SACK, FACK
13 other network programming paradigms, review

Network layers

- Physical/data link
  - Ethernet, checksums, encapsulation, CSMA/CD
  - Transmission and propagation delay
- Network layer
  - IP, datagrams, routing, RTT, addressing, ICMP, TTL, fragmentation
- Transport layer
  - UDP
  - TCP
  - Flow control, congestion and loss
- Application layer
  - BSD sockets
  - Ports and services
  - Network tools

Key papers:
Clark, *Internet Protocol Design*
Jacobson, *Congestion*
Floyd, *Tahoe…SACK*

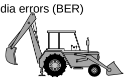
RFCs: 791, 768, 793, 1323, 2581
Text: chapters 1, 2, 3, 11, App. A

Concept Collection

- **ACK/NAK cumulative ACK**
- **ACK clocking**
- **AIMD**
- **Bandwidth-delay product**
- **Best effort**
- **Bit error rate**
- **Checksums**
- **Client/server/concurrent/iterative**
- **Congestion control/avoid**
- **Conservation of packets**
- **CIDR**
- **CSMA/CD**
- **cwnd/sstrhesh**
- **Datagram vs reliable stream**
- **Dup threshold**
- **Exponential backoff**
- **Flow control**
- **Forward ACK**
- **fragmentation**
- **Layers/encapsulation**
- **Maximum segment lifetime(MSL)**
- **MTU MSS/MTU discovery**
- **Network mask**
- **Packet switching vs circuit-based**
- **Partial ACK**
- **promiscuous**
- **Routing**
- **RTT and RTT estimation**
- **Selective ACK (SACK)**
- **Self-clocking**
- **Sliding window**
- **Slow-start**
- **Subnets/supernets**
- **Switch vs hub**
- **TTL**

Things that slow us down …

- **Physical layer**
  - Loose connectors
  - RF interference
  - Collisions
  - Slow media or media errors (BER)
  - Speed of light
  - Backhoe

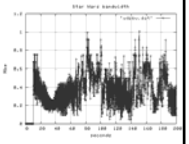


- **Link layer**
  - Half/full duplex mismatch
  - CRC errors
  - Exponential backoff
  - Packet reordering
  - NIC queues (txquelen)
  - Device (NIC) Driver software
    - interrupts
- **Network layer**
  - Fragmentation
  - Long routes
  - Slow links
  - Congestion
  - queue overflows (drops)
  - Synchronous routing updates?
  - Packet reordering (route/Juniper)
  - Software implementations/bugs
  - Firewalls/encryption
    - Block ports, ICMP
    - Examine/modify packets

Encapsulation overhead:
just handling all the layering
extra bits in headers

Things that slow us down … UDP

- Transport layer (UDP)
  - – Some UDP applications (streaming) do not backoff under heavy network load, hurting the other transport protocol (TCP) – not “TCP-friendly”
    - RealPlayer audio: 10 pkts/sec (rate-based) 70 kbs
      - – 100 users, 7 mbs → 70% of 10mbs ethernet
    - Star Wars mpeg streaming video 400 kbs
  - – DNS lookups can slow a network application
  - – Hackers use UDP to flood the network (denial of service)
- Sending a packet to a remote host
  1. ARP for local DNS server (IP address in /etc/resolv.conf)
  2. Send DNS query to local DNS (this could take a while)
  3. ARP for subnet router
  4. Send one or more packets to remote via subnet router and then out into the Internet …



Things that slow us down … TCP

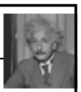
- SNDBUF limits
- RCVBUF limits
- NIC speed or bottleneck link speed
- Slow-start, delayed ACK, Nagle
- Packet loss and congestion
  - – Recovery method (Tahoe, Reno, NewReno, SACK)
- Packet reordering
- Application “protocol”

- Recovery rate sensitive to RTT (speed of light) and MSS

Our tool set

- ping/traceroute
- ifconfig/netstat
- strace
- lsof
- dig
- ethereal tcpdump/tcptrace/xplot
- ttcp/iperf/netperf

Things you might want to know for the midterm



- Concept collection
- Name the animals in the ACK zoo (cumulative, delayed, dup, selective…)
- Fragmentation/MSS/MTU discovery
- Parsing hex tcpdump of IP/TCP/UDP headers (TCP options)
- How to make a reliable stream
- TCP evolution (cwnd/ssthresh)
- Bandwidth-delay product
- The role of RTT and MSS in TCP performance
- Traceroute and TTL
- Things that go bump in the net
- Bytes/packets to send one byte of data for UDP or TCP
- Socket options for setsockopt()
- TCP open and close handshakes
- Flow control vs congestion control
- Header checksum semantics
- Socket functions that “block”
- Purpose/uses for ICMP, UDP, TCP
- Nagle/Silly Window Syndrome

Next time …

- In class MIDTERM
  - – Open book, open notes, closed mouth
- This afternoon powerpoint versions of class lectures will be in ~dunigan/ipp05/lectures.zip