Socket Programming (C/Java)

CS587x Lecture 3
Department of Computer Science
Iowa State University

Internet Socket

- Socket provides an interface to TCP/UDP
  - Allow programs to write networking applications
- Socket types of interest
  - SOCK_STREAM
    - Maps to TCP in the AF_INET family
  - SOCK_DGRAM
    - Maps to UDP in the AF_INET family

Client-Server Architecture

- Client requests service from server
- Server responds with sending service or error message to client

Simple Client-Server Example

- Client
  - socket()
  - bind()
  - listen()
  - accept()
  - send()
  - recv()
  - close()
- Server
  - socket()
  - bind()
  - listen()
  - accept()
  - send()
  - recv()
  - close()
Example: Client Programming

- Create stream socket (\texttt{socket()})
- Connect to server (\texttt{connect()})
- While still connected:
  - send message to server (\texttt{send()})
  - receive (\texttt{recv()}) data from server and process it

Initializing Socket

- Getting the file descriptor
  ```c
  int cSock;
  if ((cSock = socket(AF_INET, SOCK_STREAM, NULL)) < 0)
  {
    perror("socket");
    printf("Failed to create socket\n");
    abort();
  }
  ```

Connecting to Server

- struct hostent *host = gethostbyname(argv[1]);
- unsigned int svrAddr = *(unsigned long *) host->h_addr_list[0];
- unsigned short svrPort = atoi(argv[2]);
- struct sockaddr_in sin;
  memset (&sin, 0, sizeof(sin));
  sin.sin_family = AF_INET;
  sin.sin_addr.s_addr =svrAddr;
  sin.sin_port = htons(svrPort);
- if (connect(cSock, (struct sockaddr *) &sin, sizeof(sin)) < 0)
  {
    fprintf(stderr, "Cannot connect to server\n");
    abort();
  }

Sending Packets

- int send_packets(char *buffer, int buffer_len)
  ```c
  {
    sent_bytes = send(cSock, buffer, buffer_len, 0);
    if (send_bytes < 0)
    {
      fprintf(stderr, "Cannot send.\n");
      return 0;
    }
  }
  ```

- Needs socket descriptor,
- Buffer containing the message, and
- Length of the message
Receiving Packets

```c
int receive_packets(char *buffer, int bytes)
{
    int received = 0;
    int total = 0;
    while (bytes != 0)
    {
        received = recv(cSock, buffer[total], bytes);
        if (received == -1) return -1;
        if (received == 0) return total;
        bytes = bytes - received;
        total = total + received;
    }
    return total;
}
```

Example: Server Programming

- create stream socket (`socket()`)
- Bind port to socket (`bind()`)
- Listen for new client (`listen()`)
- User connects (`accept()`)
- Data arrives from client (`recv()`)
- Data has to be send to client (`send()`)

Server Programming

```c
struct hostent *host = gethostbyname(argv[1]);
unsigned int svrAddr = *(unsigned long *) host->h_addr_list[0];
unsigned short svrPort = atoi(argv[2]);
struct sockaddr_in sin;
memset(&sin, 0, sizeof(sin));
sin.sin_family = AF_INET;
sin.sin_addr.s_addr = svrAddr;
sin.sin_port = htons(svrPort); /* network byte order (big-endian) */
int svrSock = socket(AF_INET, SOCK_STREAM, 0);
if (bind(svrSock, (struct sockaddr *)&sin, sizeof(sin)) < 0)
{
    fprintf(stderr, "Cannot bind to network\n");
    abort();
}
listen(svrSock, 5); /* maximum 5 connections will be queued */
while (1)
{
    int cltSock = accept(svrSock, (struct sockaddr *)&cli_addr, &clilen);
    /* launch a new thread to take care of this client connection */
    /* cli_addr contains the address of the connecting client */
    /* cli_len is the buffer length that is valid in cli_addr */
    /* both cli_addr and cli_len are optional */
}
```
Java Socket Programming

- **TCP stream**
  - java.net.Socket
  - java.net.ServerSocket
- **UDP packet**
  - java.net.DatagramPacket
  - java.net.DatagramSocket

**Socket**

- java.net.Socket is used by clients to make a bi-directional connection with server
- **Socket constructors**
  - Socket(String hostname, int port)
  - Socket(InetAddress addr, int port)
  - Socket(String hostname, int port, InetAddress localAddr, int localPort)
  - /* specify a specific NIC and port to use */
  - Socket(InetAddress addr, int port, InetAddress localAddr, int localPort)
- **Creating socket**
  
  Socket csweb = new Socket("www.cs.iastate.edu", 80);

**Socket Input & Output**

```java
try {
    String s;
    Socket socket = new Socket("www.cs.iastate.edu", 80);
    BufferedReader reader = new BufferedReader(
        new InputStreamReader(socket.getInputStream()));
    PrintStream pstream = new PrintStream(socket.getOutputStream());
    pstream.println("GET /");
    while ((s = reader.readLine()) != null) {
        System.out.println(s);
    }
} catch (Exception e) {
    System.err.println("Error: "+ e);
}
```

- Socket() attempts to connect the server immediately
- Cannot set or change remote host and port
- Socket constructors may block while waiting for the remote host to respond
Some Socket Options

- `void setReceiveBufferSize()`
- `void setSendBufferSize()`
- `void setTcpNoDelay()`
- `void setSoTimeout()`

ServerSocket

- ServerSocket is used by server to accept client connections
- ServerSocket constructor
  - `public ServerSocket(int port)`
  - `public ServerSocket(int port, int backlog)`
  - `public ServerSocket(int port, int backlog, InetAddress networkInterface)`
- Creating a ServerSocket
  - `ServerSocket ss = new ServerSocket(80, 50);`
- A closed ServerSocket cannot be reopened

A Simple Server

```java
try {
    ServerSocket ss = new ServerSocket(2345);
    Socket s = ss.accept();
    PrintWriter pw = new PrintWriter(s.getOutputStream());
    pw.println("Hello There!");
    pw.println("Goodbye now.");
    s.close();
} catch (IOException ex) {
    System.err.println(ex);
}
```

Sending UDP Datagrams

1. Convert the data into byte array.
2. Create a DatagramPacket using the array.
3. Create a DatagramSocket using the packet and then call `send()` method.

Example

```java
InetAddress dst = new InetAddress("cs.iastate.edu");
String s = "This is my datagram packet";
byte[] b = s.getBytes();
DatagramPacket dp = new DatagramPacket(b, b.length, dst, 2345);
DatagramSocket sender = new DatagramSocket();
sender.send(dp);
```

Note: DatagramPacket object can be reused (e.g., setting different dst and port).
Receiving UDP Datagrams

1. Construct an empty DatagramPacket (with a buffer)
2. Pass the object to a DatagramSocket (with a port)
3. Call the DatagramSocket's receive() method
4. The calling thread blocks until a datagram is received

```
byte buffer = new byte[1024];
DatagramPacket incoming = new DatagramPacket(buffer, buffer.length);
DatagramSocket ds = new DatagramSocket(2345);
ds.receive(incoming);
```

```
byte[] data = incoming.getData();
String s = new String(data, 0, incoming.getLength());
System.out.println("Port " + incoming.getPort() + 
   " on " + incoming.getAddress() + 
   " sent this message: ");
System.out.println(s);
```

A Mistake You Want to Avoid

```
byte[] buf = new byte[1024];
DatagramPacket incoming = new DatagramPacket(buf, buf.length);
DatagramSocket ds = new DatagramSocket(2345);
for (; ;)
{
ds.receive(incoming);
byte[] data = incoming.getData();
    System.arraycopy(incoming.getData(), 0, data, 0, data.length);
    new DataProcessor(data).start();
}
```

Correct Way

```
byte[] buf = new byte[1024];
DatagramPacket incoming = new DatagramPacket(buf, buf.length);
DatagramSocket ds = new DatagramSocket(2345);
for (; ;)
{
ds.receive(incoming);
    byte[] data = new byte[incoming.getLength()];
    System.arraycopy(incoming.getData(), 0, data, 0, data.length);
    new DataProcessor(data).start();
}
```

Redundant Array of Inexpensive Disk

**RAID Management**
- Monitor the health condition of RAID subsystems
- Report any failure instantly
- Provide disaster recovery
  - array rebuild, spare disks reassign, etc.
Remote and Centralized Storage Management

Client Design: Two Threads

- **BeaconSender**: Send the following message to server every minute using UDP datagram.

  ```
  struct BEACON {
    int ID; // randomly generated during startup
    int StartUpTime; // the time when the client starts
    char IP[4]; // the IP address of this client
    int CmdPort; // the client listens to this port for cmd
  }
  ```

- **CmdAgent**: Receive and execute remote commands and send results back using TCP socket. You implement two commands:

  1. `void GetLocalOS(char OS[16], int *valid)`
     - `OS[16]` contains the local operation system name
     - `valid = 1` indicates OS is valid
  2. `void GetLocalTime(int *time, int *valid)`
     - `time` contains the current system clock
     - `valid = 1` indicates time is valid

Server Design

- **BeaconListener thread**
  - Receive beacons sent by clients
  - For each new client, spawn a thread called ClientAgent

- **ClientAgent(beacon) thread**
  - Send command GetLocalOS() to the corresponding client
  - Get the result back and display the OS
  - Send command GetLocalTime() to the corresponding client
  - Get the result back and display the execution time

Homework #1

- (UDP) Send beacon every minute
- (TCP) Send command and get result back

Server (C code)
- Client (java code)