Socket Programming

Outline:
- Socket programming introduction.
- Socket programming with TCP and UDP.
- References and assignments.
- UNIX networking tools.

Socket Programming (cont.)

Socket: the interface between application process and end-to-end transport protocol (TCP or UDP)

Port Numbers

- Multiple sockets might exist in each host.
- A port number identifies each such socket in each host.
- Each port number is a 16-bit number, ranging from 0 to 65535.
- Port numbers ranging from 0 to 1023 are called well-known port numbers and are restricted.

Socket Programming with TCP

Client must contact server
- server process must first be running
- server must have created socket (door) that welcomes client’s contact
Client contacts server by:
- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket:
  client TCP establishes connection to server TCP
- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients

Socket API

- Introduced in BSD 4.1 UNIX, 1981
- Explicitly created, used, and released by apps
- Client-server paradigm
- Two types of transport service via socket API:
  - reliable, byte stream oriented (TCP)
  - unreliable datagram (UDP)

A host-local, application-created, OS-controlled interface (a “door”) into which application process can both send and receive messages to/from another application process.

TCP provides reliable, in-order transfer of bytes (“pipe”) between client and server.
Stream Jargon

- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, e.g., keyboard or a TCP socket.
- An output stream is attached to an output source, e.g., monitor or a TCP socket.

Client/Server TCP Socket Interaction: An Example

Example client-server app:
1) client reads line from standard input (inFromUser stream), sends to server via socket (outToServer stream)
2) server reads line from socket
3) server converts line to uppercase, sends back to client
4) client reads, prints modified line from socket (inFromServer stream)

Client/Server TCP Socket Interaction: Pseudo-code

Server (running on hostid)
- create socket, port=x
- bind, connect to hostid
- listen (max clients)
- accept (new client conn)
- read/write

Client
- connect to server
- read/write

Basic Socket API in C

- Create a socket
  `int socket(int domain, int type, int protocol)`
  - domain = PF_INET, PF_UNIX
  - type = SOCK_STREAM, SOCK_DGRAM

- Passive open a socket connection (server-side)
  `int bind(int socket, struct sockaddr * addr, int addr_len)`
  `int listen(int socket, int backlog)`
  `int accept(int socket, struct sockaddr * addr, int addr_len)`

Basic Socket API in C (cont.)

- Active open a socket connection (client-side)
  `int connect(int socket, struct sockaddr * addr, int addr_len)`

- Send/receive messages
  `int send(int socket, char * msg, int mlen, int flags)`
  `int recv(int socket, char * buf, int blen, int flags)`

- Close a socket connection
  `int close(int socket)`

Socket Programming with UDP

UDP: no "connection" between client and server
- no handshaking
- sender explicitly attaches IP address and port of destination to each packet

UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

Application viewpoint:
- UDP transmitted data may be received out of order, or lost
Client/Server UDP Socket Interaction: An Example

Output:
sends packet (TCP sent "byte stream")

Input:
receives packet (TCP received "byte stream")

Client
process

Sends packet to
network transport

UDP
socket

Client/Server UDP Socket Interaction: Pseudo-code

Server (running on hostid)

create socket, port=x for incoming request:
socket = DatagramSocket()

read request from serverSocket

write reply to serverSocket

Client

create socket, port=x

for incoming request:

serverSocket = DatagramSocket()

read reply from serverSocket

Client/Server UDP Socket Interaction: Pseudo-code continued

close clientSocket

References/Tutorials on Socket Programming

C-language:

• Reference: UNIX manual pages.

Java-language:

• Reference:
  • http://java.sun.com/products/jdk/1.2/docs/api/
  • Tutorial: http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html

Assignment #0 – Warm-up on Socket Programming

• Will not be graded. You are advised and expected to finish it by this Sunday.
• Write simple TCP programs in Java and C.
• The server reads an input text string from any client, modifies it slightly and returns back to the client.
• The client reads an input string from keyboard, sends it to the server. After the server responds, it displays the response.
• Four programs: client in C, server in C, client in Java, server in Java.
• What happens if a C client contacts a Java server?
• Additional work:
  – do this with UDP sockets.
  – do this using Perl.

Assignment #1 – Building A Multi-threaded Web Server

For each HTTP request, the server does the following:

• accepts the request
• parses header
• obtains requested file from server’s file system
• creates HTTP response message:
  – header lines + file
  – sends response to client
• multi-threading is simple
• after creating server, you can request file using a browser (e.g. IE or Netscape)
• details coming out over this weekend

UNIX Networking Tools

• ping – telling if a server is alive
  ping tallinn.cs.rochester.edu
• traceroute – find intermediate nodes to a destination
  traceroute booboo.cs.ucsb.edu
• nslookup – query domain name servers (DNS); find the IP address of a machine
  nslookup tallinn.cs.rochester.edu
• netstat – show various network status of a host; find active TCP/UDP sockets and their states
  netstat -a
UNIX Networking Tools (cont.)

Find the number of active sockets for a particular process through /proc filesystem in Linux:
- find the process ID of the process you are interested in (through ps), e.g., 20417
- "cd /proc/20417/fd"
- "ls -l"
- "/proc/20417/" contains runtime information about this process

Disclaimer

- Parts of the lecture slides contain original work of James Kurose, Larry Peterson, and Keith Ross. The slides are intended for the sole purpose of instruction of computer networks at the University of Rochester. All copyrighted materials belong to their original owner(s).