Sockets Programming

CS144 Review Session 1
April 4, 2008
Ben Nham
Announcements

• New classroom for lecture
  – 200-034, MW 4:15-5:30
• Simple client assignment due Wednesday, 4/9
• Use newsgroup for general questions
  – su.class.cs144
  – Refresh your newsgroup list from the server if you can’t find it
Simple Client Overview

• Opens TCP socket to other host
  – Does DNS lookup if necessary

• Reads request from command line, appending "\r\n", and sends it through the socket

• Echoes response to stdout

• Demo: Sending HTTP request to SCS homepage
  – ./sc www.scs.stanford.edu 80 "GET /"
Sockets and TCP/IP

- In TCP/IP:
  - Endpoint has unique (TCP port, IP address) pair
  - Connection between two endpoints is identified by the pair \([(IP, port)_{src}, (IP, port)_{dst}]\)
- All Unix I/O streams are referenced by descriptors
  - Socket maps a descriptor to an endpoint
  - Connecting sockets allows us to connect endpoints and do I/O
Socket API for Client

- **socket**
  - `int socket(int domain, int type, int protocol)`
  - Returns a descriptor associated with a new endpoint

- **bind**
  - `int bind(int sd, struct sockaddr *addr, u_int addr_len)`
  - Set addr/port of endpoint for socket descriptor `sd`
  - Optional for client (lets the kernel choose some available port with the default IP address)

- **connect**
  - `int connect(int sd, struct sockaddr *addr, u_int addr_len)`
  - Connect to destination address + port endpoint

- **send/recv**
  - `int send(int sd, void *buf, int len, int flags)`
  - `int recv(int sd, void *buf, int len, int flags)`
  - Two-way communication

- **shutdown**
  - `int shutdown(int sd, int how)`
  - Partial or complete connection teardown
Sockets API for Server

int socket(int, int, int)
int bind(int, struct sockaddr *, u_int)
int listen(int sd, int backlog)
  • Wait for a client to connect to this port
int accept(int sd, struct sockaddr *addr, u_int *addr_len)
  • Accept connection, returning a new descriptor for this (IP, port)_{src} – (IP, port)_{dst} pair
int send(int, void *, int)
int recv(int, void *, int)
int shutdown(int, int)
Typical Request-Response Client-Server Pair

Client
- socket
- bind
- connect
- send/recv
- shutdown

Server
- socket
- bind
- listen
- accept
- send/recv
- shutdown

Request:
"GET /

"Can I connect?"

"Yes. Can I connect?"

"Yes."

Response: "<html>...

One side starts shutdown
Typical Request-Response Client-Server Pair

Client

- socket
- bind
- connect
- send/recv
- shutdown

Server

- socket
- bind
- listen
- accept
- send/recv
- shutdown

Client

1. Connect
2. Send client request: "GET /
3. Receive server response: "<html>..."
4. One side starts shutdown
Example Daytime Server

- See the posted daytime.c

- Problems
  - Doesn’t check return values of system calls
    - You should check the return value and use perror() or fprintf(stderr, strerror(errno)) to print out an informative error message
  - Doesn’t handle multiple clients simultaneously
  - Problems with re-using the same port (use setsockopt—covered in next review session)
Main Structures

• Generic socket address
  struct sockaddr {
      u_short sa_family;
      char sa_data[14];
  };

• TCP/UDP + IPv4 specific address – convenience parallel struct for sockaddr
  struct sockaddr_in {
      u_short sa_family;       // usually AF_INET
      u_short sin_port;
      struct in_addr sin_addr; // see below
      char sin_zero[8];        // zero out
  };

• IP Address
  struct in_addr {
      u_long s_addr;           // Network byte order
  };

Next two slides by Clay Collier
Useful Functions

```c
struct hostent *gethostbyname(const char *name)
```
- Converts domain names and dotted-quad IP addresses into numerical IP addresses via DNS

```c
struct servent *getservbyname(const char *name, const char *proto)
```
- Query /etc/services to find expected protocol and port for a service
- Example: getservbyname("http", NULL) to find it resides on tcp port 80

`getsockname(...)`, `getpeername(...)`
- Gets IP address and port for source/destination
More Notes

• Partial sends and receives
  – Receive might not return with all the bytes requested

• Endian issues
  – All shorts/integers going over the wire must be encoded using htons/htonl
  – All shorts/integers being read from the wire must be decoded using ntohs/ntohl
  – Why don’t we have to worry about endianness for text? For arbitrary binary data?
Grading

• No set rubric yet
• Mostly functionality
  – Test yourself by sending (for example) http requests
• Some style points
  – Don’t write everything in main
  – Handle partial receives and other edge cases
  – Check return values of system calls
Resources

• IPC tutorial
• Man pages
• Outside references
  – Beej’s Sockets Tutorial: [http://beej.us/guide/bgnet/](http://beej.us/guide/bgnet/)
  – *Unix Network Programming* by Stevens
• Newsgroup
• Office hours