Announcements

• Today’s plan:
  – Finish up HTTP
  – DNS/naming
  – Socket options
  – Signals and timeouts
Part 0: Finish up HTTP
HTTP Responses

- Response Messages
  - Like request messages, response messages begin with a single START LINE.
  - In this case, the line specifies the version of HTTP being used, a three-digit code indicating whether or not the request was successful, and a text string giving the reason for the response.

- Example:
  
  HTTP/1.1 200 OK
  Content-Type: text/html
  Content-Length: 291
borabora:~ gmimporter$ telnet oec-vmweb09.ucsd.edu 80
Trying 132.239.8.67...
Connected to oec-vmweb09.ucsd.edu.
Escape character is '^[].

GET /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 12 Jan 2015 19:36:37 GMT
Server: Apache/2.2.22 (Ubuntu)
Last-Modified: Thu, 28 Feb 2013 17:35:36 GMT
ETag: "fc7b21-a-4d6cc51858aec"
Accept-Ranges: bytes
Content-Length: 10
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

It works!

Connection closed by foreign host.
borabora:~ gmimporter$
HTTP Response Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Example Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xx</td>
<td>Informational</td>
<td>request received, continuing process</td>
</tr>
<tr>
<td>2xx</td>
<td>Success</td>
<td>action successfully received, understood, and accepted</td>
</tr>
<tr>
<td>3xx</td>
<td>Redirection</td>
<td>further action must be taken to complete the request</td>
</tr>
<tr>
<td>4xx</td>
<td>Client Error</td>
<td>request contains bad syntax or cannot be fulfilled</td>
</tr>
<tr>
<td>5xx</td>
<td>Server Error</td>
<td>server failed to fulfill an apparently valid request</td>
</tr>
</tbody>
</table>

Five types of HTTP result codes

200: OK
400: Client Error
   403: Not allowed
   404: Not found
Traditional Applications

• World Wide Web
  – TCP Connections
    • The original version of HTTP (1.0) established a separate TCP connection for each data item retrieved from the server.
    • It’s not too hard to see how this was a very inefficient mechanism: connection setup and teardown messages had to be exchanged between the client and server even if all the client wanted to do was verify that it had the most recent copy of a page.
    • Thus, retrieving a page that included some text and a dozen icons or other small graphics would result in potentially dozens of separate TCP connections being established and closed.
HTTP 1.1

- TCP Connections
  - To overcome this situation, HTTP version 1.1 introduced *persistent connections*—the client and server can exchange multiple request/response messages over the same TCP connection.
  - Persistent connections have many advantages.
    - First, they obviously eliminate the connection setup overhead, thereby reducing the load on the server, the load on the network caused by the additional TCP packets, and the delay perceived by the user.
    - Second, because a client can send multiple request messages down a single TCP connection, TCP’s congestion window mechanism is able to operate more efficiently.
      » This is because it’s not necessary to go through the slow start phase for each page.
HTTP 1.1 “Persistent Connections”
Request headers

• **Host:**
  – Indicates the name of the server you are accessing
  – Used to implement virtual hosts

• **User-Agent:**
  – Identifies what software is issuing the request
  – E.g.:
    • Opera/9.25 (Windows NT 6.0; U; en)
    • Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/125.2 (KHTML, like Gecko) Safari/125.8
Response headers

• Server:
  – Identifies the server
    • Server: Apache/2

• Content-Length:
  – How many octets in the response

• Content-Type:
  – text/html
  – image/jpg
  – image/png
Demo 0: HTTP headers

• You can follow along at
  – [http://cseweb.ucsd.edu/~gmporter/classes/wi17/cse124/assets/project1/htdocs/index.html](http://cseweb.ucsd.edu/~gmporter/classes/wi17/cse124/assets/project1/htdocs/index.html)

• Usage:
  curl –v http://...
Part 1: Domain Name System (DNS)
Overview

- [www.cs.ucsd.edu](http://www.cs.ucsd.edu) → 132.239.8.67
- 1982: single hosts.txt file stored and distributed from a central site
- Contained all hostname to IP address mappings
- Centralized control did not fit with distributed management
- Number of hosts changed from number of timesharing systems to number of workstations
  - Organizations to users
  - Exponential resource usage for distributing the file
Domain Name System

• Hierarchical namespace with typed data
• Control delegated in hierarchical fashion
  – Convince node above you to delegate control
• Designed to be extensible w/support for new data types
• 1985: some hosts solely utilize DNS
Hierarchical Design
Domain Name System (DNS)

• Translate human understandable names to machine understandable names
  – E.g., www.cs.ucsd.edu ➔ 132.239.8.67
  – Hierarchical structure
    • Every DNS server knows where the “root” is
    • The root can tell you how to get to .edu
    • .edu server can tell you how to find ucsd.edu
    • ucsd.edu tells you about cs.ucsd.edu
    • cs.ucsd.edu translates www.cs.ucsd.edu ➔ 132.239.8.67
  – Caching along the way to improve performance
Root name servers
Query Processing

• Query local name server
  – Authoritative/cached answers

• Support both recursive and iterative queries

• If not cached locally, locate server lowest in the hierarchy with entry in local DB
  – In the worst case, contact root (.)
  – Cache locally with TTL
Zones and Caching

• Mechanisms for data distribution

• Zones
  – Provide local autonomy
  – Any contiguous set of nodes in the tree
  – Can be grown to arbitrary size
  – Each domain should provide redundant servers

• Caching
  – Time to live (TTL) associated with each name
    • low value => higher consistency
    • high value => better performance (less traffic)
DNS Lookup Example

- Client
- Local DNS proxy
- www.cs.ucsd.edu
  - cs.ucsd.edu
  - ucsd=IPaddr
  - cs=IPaddr'
  - www=IPaddr'
- edu DNS server
- ucsd DNS server
- cs DNS server
Mapping names to addresses

NAME

getaddrinfo, freeaddrinfo, gai_strerror - network address and service translation

SYNOPSIS

#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints, 
                struct addrinfo **res);

void freeaddrinfo(struct addrinfo *res);

const char *gai_strerror(int errcode);
Linked list of ‘addrinfo’ structs

struct addrinfo {
    int ai_flags;
    int ai_family;
    int ai_socktype;
    int ai_protocol;
    socklen_t ai_addrlen;
    struct sockaddr *ai_addr;
    char *ai_canonname;
    struct addrinfo *ai_next;
};

• Q: Why a linked list?
• Q: Which of the multiple results should you use?
Hints

• Can provide hints as to what you’re looking for:
  – Server socket (hints.ai_flags = AI_PASSIVE)
    • Returned sockaddr_in suitable for server-side bind()
  – Client socket (otherwise)
  – IPv4 vs. IPv6
  – TCP vs. UDP
Part 2: Socket options
Socket options: motivation

• Basic “out of the box” socket functionality fine for most purposes
  – But what if you need to tweak the behavior?

• Can set/get ‘options’ on sockets

• These options apply to different layers of the network stack:
  – IP
  – TCP
  – Socket
Interesting options

• Send and receive buffer sizes
  – What is the default?

  [gmpporter@seed-f60-100 ~]$ cat /proc/sys/net/ipv4/tcp_rmem
  4096  87380  6291456
  [gmpporter@seed-f60-100 ~]$ cat /proc/sys/net/ipv4/tcp_wmem
  4096  16384  4194304

  Minimum  Default  Maximum

• Can we change that value?
  – Yes!
Setting/getting socket options

GETSOCKOPT(2)  Linux Programmer's Manual  GETSOCKOPT(2)

NAME

getsockopt, setsockopt – get and set options on sockets

SYNOPSIS

#include <sys/types.h>    /* See NOTES */
#include <sys/socket.h>

int getsockopt(int sockfd, int level, int optname,
                void *optval, socklen_t *optlen);
int setsockopt(int sockfd, int level, int optname,
               const void *optval, socklen_t optlen);

<table>
<thead>
<tr>
<th>Level</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOL_SOCKET</td>
<td>SO_SNDBUF</td>
<td>Send buffer size</td>
</tr>
<tr>
<td></td>
<td>SO_REUSEADDR</td>
<td>Allow TCP port to be reused immediately</td>
</tr>
<tr>
<td></td>
<td>SO_RCVTIMEO</td>
<td>Set a recv() timeout</td>
</tr>
<tr>
<td></td>
<td>SO_SNDTIMEO</td>
<td>Set a send() timeout</td>
</tr>
</tbody>
</table>
Reusing sockets

int optval = 1;

/* enable sockets to be immediately reused */
if (setsockopt(serv_sock, SOL_SOCKET, 
   SO_REUSEADDR, &optval, sizeof(optval)) != 0)
{
    die_system("setsockopt() failed");
}
Part 3: Signals and timeouts
Signals

- OS mechanism to asynchronously interrupt a program
- Why is this useful?
  - Kill a runaway/hung process
  - Notify program that there is activity on the keyboard
  - Disk read operation has completed
  - The dreaded SIGSEGV
Signals in action

- $ sleep 9999999
- How to stop this program?
Signals and networking

• Signals can be used to implement *timeouts*

• Examples:
  – Close connection after 3 minutes of inactivity
  – HTTP server: is the client going to send another request? Set timeout for e.g., 5 seconds

• Useful any time you need to stop blocking
  – recv()
  – send()
  – ...

Signals

- SIGALRM
  - Issued after a set period of time goes by
  - Like an alarm clock for your program
- Others in D&C Chapter 6.2
Setting up event handlers

SIGACTION(2) Linux Programmer's Manual SIGACTION(2)

NAME

sigaction – examine and change a signal action

SYNOPSIS

#include <signal.h>

int sigaction(int signum, const struct sigaction *act,
             struct sigaction *oldact);

struct sigaction {
    void    (*sa_handler)(int);
    void    (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int     sa_flags;
    void    (*sa_restorer)(void);
};

Function to handle event

How to handle other events during the handling of this event

Rest of fields can be set to NULL/0
So how do we use this?

• Define the event handling function
  – void myfun(int signal);

• Associate that function with the signal you want to handle
  – sigaction() call
What does ‘mask’ mean?

• Signals arrive unpredictably and asynchronously
  – Get a SIGINT or SIGTERM for example

• What happens if, if your handler for SIGINT, another SIGINT comes?

• Can simplify our handler by ‘masking’ signals during our event handler
  – Helper functions provided (e.g., sigfillset(...))
Signals and networking APIs

• What happens to a blocking call when an event comes in?
  – Control transferred to event handler
  – When control returned, the blocking stops, and an error code is returned

• Recv()
  – Might return fewer bytes than requested, or EINTR return code if no bytes received

• Send()
  – Might send fewer bytes than requested or EINTR if no bytes sent
Demo 2

Setting a receiver timeout on our echo server using an event handler, closing the connection after 3 seconds of client inactivity
Demo 2 overview

1. Define our event handler
2. Setup the event handler with sigaction()
3. Change our recv() code to check for EINTR return code
   1. If so, close the connection
Alternative timeout mechanism

• Instead of a SIGALRM event handler, can set timeouts using a socket option

• Why might this be a better option in some cases?