Announcements

• Today’s plan:
  – Signals and timeouts
  – DNS/naming
Part 1: Signals and timeouts
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()
  - ...
Two approaches

• SIGALRM
  – Issued after a set period of time goes by
  – Like an alarm clock for your program

• Other signals 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 SIGTERMT 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?
Demo 3

Setting a receiver timeout on our echo server with socket options, closing the connection after 3 seconds of client inactivity
Demo 3 overview

1. Set the timeout value with setsockopt()
2. Change recv() calls to check for the timeout
   – EWOULDBLOCK instead of EINTR
   – Otherwise the same as the alarm example
Part 2: Domain Name System (DNS)
Overview

- [www.cs.ucsd.edu](http://www.cs.ucsd.edu) → 132.239.8.67
- 1974: single hosts.txt file stored and distributed from a single site: Stanford University
  - 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
  - Email made the problem worse
Domain Name System

• 1982: proposal for decentralized directory—DNS
• 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

www.cs.ucsd.edu

client

dns proxy

cs.ucsd.edu
ucsd=IPaddr

cs=IPaddr'

edu DNS server

ucsd DNS server

cs DNS server
Mapping names to addresses

GETADDRINFO(3) Linux Programmer's Manual GETADDRINFO(3)

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

```c
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