Lesson of the day 1:
Always backup your computer

Lesson of the day 2:
That backup needs to be automatic!
Announcements

• HWs 2 and 3 posted
• Asking ACMS to upgrade compilers

• Today:
  – Clarify HW2 specification
  – Finish up client/server echo client
  – DNS overview
  – Project 1, HWs 2 & 3 Q&A
HW2 clarification

• Server changes:
  – Takes param N indicating how many times to replicate the string

• Client changes:
  – Sends string, reads response to figure out N

• Why?
  – Practice reading data when you don’t know length ahead of time
  – Practice looking for a delimiter (i.e., NULL)

• Client adds a NULL after the string
• Server replies with N concatenated strings (doesn’t add NULLs)

• Example:
  – Client → Server
    • “Hello<0>” (N=3)
  – Server → Client
    • “HelloHelloHello”
Recall: Server overview

Steps
1. Handle command line arguments
2. Create network socket
3. Bind socket to an interface
4. Tell the socket to listen for incoming connections
5. Accept an incoming connection:
   – Read/write to the socket
6. Close the socket

Socket API used
1. n/a
2. socket()
3. bind()
4. listen()
5. accept()
6. send/recv()
7. close()
Step 3: bind

• We need to tell the socket what IP address and port to listen for incoming connections

• `int bind(int sockfd, const struct sockaddr *addr, socklen_t len);`

• We don’t care which IP address to bind to
  – `servAddr.sin_addr.s_addr = htonl(INADDR_ANY);`

• We do care about the listening port
  – `servAddr.sin_port = htons(servPort);`
Step 4: listen

• Tells the OS that this socket is going to be a server socket
  – E.g., that it should listen for incoming commands

• `int listen(int sockfd, int backlog);`
  – ‘backlog’ specifies how many pending connections can exist before additional ones are refused
Step 5: Accepting new connections

- A “listening” socket isn’t actually used to send/receive data...
  - Otherwise e.g., a web server could only handle one client at a time
  - Because port 80 (www) would be “used up” by a single client

- Instead, listening socket used to accept a new connection, which spawns a new per-connection ‘client’ socket
Server socket vs. Client socket

ACCEPT(2) 

NAME
accept, accept4 — accept a connection on a socket

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

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
recv() call

NAME
recv, recvfrom, recvmsg – receive a message from a socket

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

ssize_t recv(int sockfd, void *buf, size_t len, int flags);

RETURN VALUE
These calls return the number of bytes received, or -1 if an error occurred. In the event of an error, errno is set to indicate the error. The return value will be 0 when the peer has performed an orderly shutdown.
Domain Name System (DNS)
Overview

• 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

- root
  - org
  - mil
  - edu
  - com
  - uk
  - ca
  - gwu
  - ncsu
  - ucsd
    - ece
    - cs
      - hobo
      - www
    - blink
      - ctrl
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

www.cs.ucsd.edu
Part 3: an API to DNS
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

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)
  – Client socket (otherwise)
  – IPv4 vs. IPv6
  – TCP vs. UDP