Project 3: A Bigger Network Programming Challenge

Networking
CS 3470, Section 1
Simple Talk

- Separate programs – client and server
- Server reversed the message and sent it back to the client
- But two users still can’t talk with each other…
Chat Program

- One program!
  - Combines both client and server functionality
- Two chat program users can talk to each other
- Also introduces checksums, packet structures, network formatting, and advanced argument processing
- Compiles with a Makefile
# This is a comment. Don’t use C-style comments
CC=gcc
CFLAGS=-Wall

all: chat

chat: chat.c
   $(CC) -o chat chat.c $(CFLAGS)

clean:
   rm chat
int main(int argc, char **argv)
{
    /* Are we invoking the server or client? Let's figure it out based
     * on the number of arguments. */
    if(argc > 3) /* change when implementing –c */
        client(argc, argv);
    else
        server();

    return 0;
Helpful Functions

- atoi – converts a string to an int
- strcmp or strncmp – compares two strings
- When assigning a string, don’t use =
  - Use strcpy

- For help, use the Linux man pages
Back and Forth Chatting

- Do this first!
- Both send() and rcv() must be in the client and server function's while loops

### Server (on IP address 10.42.42.1 prog1)

```
$./chat -n Trinity
Welcome to Chat!
Waiting for a connection on 10.42.42.1 port 1234
Found a friend! You receive first.
Neo: hello!
You: Hi! How are you?
Neo: I'm great.
You: That's good. Ok, bye!
^C
```

### Client (on IP address 10.42.42.2 prog2)

```
$./chat -p 1234 -s 10.42.42.1 -n Neo
Connecting to server... Connected!
Connected to a friend! You send first.
You: hello!
Trinity: Hi! How are you?
You: I'm great.
Trinity: That's good. Ok, bye!
You: 
```


Packet Formatting

- Your packet should look like this

<table>
<thead>
<tr>
<th>Version</th>
<th>Sender Username</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>(16 bits)</td>
<td>(8 bytes)</td>
<td>(144 bytes)</td>
</tr>
</tbody>
</table>
Packet Formatting

- How do I do this?
- Create a struct packet that holds all those fields
- Send the struct packet over the socket
Creating a Struct Packet

- Assign 2 to version field
- Assign the sender’s username to the username field
- Assign the data to the data field
Checksum and Version Printing

- Used in debugging mode
Other Things

- Cannot have any pointers in struct packet before sending it over the socket
  - Why?
Other Things

- Must be aware of byte ordering!
Byte Ordering

- High-order byte
- Low-order byte
- MSB
- 16-bit value
- LSB
- Increasing memory address
- Little-endian byte order
- Big-endian byte order
- Address A
- Address A+1
Implications of Byte Order

- Unfortunately there is no standard between these two byte orderings and we encounter systems that use both formats.
- We refer to the byte ordering used by a given system as **host byte order**.
- The sender and the receiver must agree on the order in which the bytes of these multi-byte field transmitted: specify **network byte order**, which is **big-endian byte ordering**.
# Byte Order Functions

```
#include <netinet.h>

/* Host to network */
uint16_t htons(uint16_t host16bitvalue)
Converts a 16-bit integer from host to network byte order

uint32_t htonl(uint32_t host32bitvalue)
Converts a 32-bit integer from host to network byte order

Both return: value in network byte order

/* Network to host */
uint16_t ntohs(uint16_t net16bitvalue)
uint32_t ntohl(uint32_t net32bitvalue)

Both return: value in host byte order
```
When do we use hton/ntoh functions?

- Use hton any data structure we send through our socket that has data units greater than one byte
  
  - E.g., if our first struct member is a 2-byte header, and sender/receiver have different memory orderings, number would look very different to each machine
Man Pages

- Use man pages to look up useful information
  - $> \text{man cat}
    - Get information about shell commands
  - $> \text{man bind}
    - Get information about C library/system calls
    - Also tells you which header files to include
  - $> \text{man man}
    - Get information about man pages
Man Pages

Sometimes you need to specify a man section

- E.g., `printf` is both a shell command and a C library call

- Use either
  - `man 1 printf`
  - `man 3 printf`

- See
  - [http://www.cs.uni.edu/~diesburg/courses/cs3470_fa14/resources/man_page_levels.htm](http://www.cs.uni.edu/~diesburg/courses/cs3470_fa14/resources/man_page_levels.htm)
C Header Files

- Located at /usr/include
- Can find out what functions are available for you to use and struct definitions
  - (Hint: check out <string.h> and <strings.h>)
Byte Manipulation Functions

```c
#include <strings.h>
/* Berkeley-derived functions */
void bzero(void *dest, size_t nbytes)
Set the first part of an object to null bytes

void bcopy(const void *src, void *dest, size_t nbytes);

int bcmp(const void *ptr1, const void *ptr2, size_t nbytes)
/* return: 0 if equal, nonzero if unequal */

#include <string.h>
/* ANSI C defined functions */
void *memset(void *dest, int c, size_t len)
Sets the first len bytes in memory dest to the value of c

void *memcpy(void *dest, const void *src, size_t nbytes)

void memcmp(const void *ptr1, const void *ptr2, size_t nbytes)
```