CSE 124
January 14, 2016

Winter 2016, UCSD

Prof. George Porter

Announcements

- HW 2 due this afternoon
- Project 1 has been posted
- Textbook is in the bookstore

- Today's plan:
  - Finish up API on DNS
  - Briefly discuss framing, encoding, and protocol design
  - Let's design a protocol!
  - Then let's implement that protocol

Part 1: 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

Demo: Chapter 3

Part 2: Encoding and framing

Material in Chapter 5

Not going to cover in class

Encoding (in one slide)

- C's 'int', 'long', … not well defined
  - 32 bits? 64 bits?

- Use 'standard' int types instead:
  - int32_t
  - int8_t
  - uint32_T
  - uint64_t
  - …

Chapter 5 stuff we're not covering in class

- Byte ordering
- Signedness, sign extension,
- Encoding integers by hand
- C struct layout, padding

Buffered streams

- FILE streams are compatible with TCP sockets
  - FILE * fdopen(int socket, const char * mode)
  - fwrite()
  - fread()
  - fflush()
  - fclose()
- Benefits:
  - They are buffered (minimize context switches)
  - They read/write fixed-length objects

Stream examples

```
FILE * out = fdopen(sock, “w”)
FILE * in = fdopen(sock2, “r”)

uint8_t val8 = 3;
If (fwrite(&val8, sizeof(val8), 1, out) != 1) …

uint64_t val64;
If (fread(&val64, sizeof(val64), 1, in) != 1)…
val64 = ntohl(val64);
```

Part 3: Protocol design

Protocols

- Structured ways to communicate information
- An art and a science

- Framing
  - How do you send and receive messages?
  - More than just ‘date’ or ‘time’

- Encoding
  - How do you interpret those messages?
  - Text? Integers? Floating point numbers? Video frames? Photos? Facebook profiles?

Framing

- Ensuring that you send/receive an entire (variable-length) message
  - Delimiter-based
  - Explicit length

Encoding/parsing

- How to interpret a message?
- Text
- Binary

Key design principle

- Separate out framing from parsing
  - Via layering
    - Layer 0: send/receive raw bytes
    - Layer 1: send/receive messages
    - Layer 2: parse/encode data structures into messages

In-class exercise prep

- Break into groups of about 5 students
  - Ensure one of you has a laptop
- Download the code linked from today's entry in the syllabus
- Make sure it compiles on your server and/or on the seed-x60-yyy server
  - This is your starting code

In-class exercise part 1

- Design a protocol to keep track of players' scores in a video game. Each player has:
  - A username
  - An ID between 0 and 20,000,000
  - Their score
  - A 256x256 pixel avatar image
- Your protocol should be able to set or get player information
  - Implicitly create the player if they don't exist

- Your group will be assigned a 'text' or 'binary' representation to develop.

In-class exercise part 2

- Code up the implementation
  - Either text or binary, as assigned