Week 11 - Wednesday

COMP 2400

Last time

- What did we talk about last time?
- Binary files
- Started low-level I/O

Questions?

Project 5

Quotes

The key to performance is elegance, not battalions of special cases. The terrible temptation to tweak should be resisted unless the payoff is really noticeable.

Jon Bently and M. Douglas McIlroy
Computer Scientists at Bell Labs

Low Level File I/O

Includes

- To use low level I/O functions, include headers as follows:

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
```

- You won't need all of these for every program, but you might as well throw them all in

open()

- To open a file for reading or writing, use the **`open()`** function
  - There used to be a **`creat()`** function that was used to create new files, but it's now obsolete
- The **`open()`** function takes the file name, an **`int`** for mode, and an (optional) **`int`** for permissions
- It returns a file descriptor

```
int fd = open("input.dat", O_RDONLY);
```

read()

- Opening the file is actually the hardest part
- Reading is straightforward with the **`read()`** function
- Its arguments are
  - The file descriptor
  - A pointer to the memory to read into
  - The number of bytes to read
- Its return value is the number of bytes successfully read

```
int fd = open("input.dat", O_RDONLY);
int buffer[100];
read( fd, buffer, sizeof(int)*100 );
```

write()

- Writing to a file is almost the same as reading
- Arguments to the **`write()`** function are
  - The file descriptor
  - A pointer to the memory to write from
  - The number of bytes to write
- Its return value is the number of bytes successfully written

```
int fd = open("output.dat", O_WRONLY);
int buffer[100];
int i = 0;
for( i = 0; i < 100; i++ )
     buffer[i] = i + 1;
write( fd, buffer, sizeof(int)*100 );
```

close()

- To close a file descriptor, call the **`close()`** function
- Like always, it's a good idea to close files when you're done with them

```
int fd = open("output.dat", O_WRONLY | O_CREAT | O_TRUNC,
0644);
// Write some stuff
close( fd );
```

Example

- Use low level I/O to write a hex dump program
- Print out the bytes in a program, 16 at a time, in hex, along with the current offset in the file, also in hex
- Sample output:

```
0x000000 : 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
0x000010 : 02 00 03 00 01 00 00 00 c0 83 04 08 34 00 00 00
0x000020 : e8 23 00 00 00 00 00 00 34 00 20 00 06 00 28 00
0x000030 : 1d 00 1a 00 06 00 00 00 34 00 00 00 34 80 04 08
```

File descriptors revisited

- A file descriptor is not necessarily unique
  - Not even in the same process
- It's possible to duplicate file descriptors
  - Thus, the output to one file descriptor also goes to the other
  - Input is similar

Duplicating descriptors on the command line

- **`stderr`** usually prints to the screen, even if **`stdout`** is being redirected to a file

```
./program > output.txt
```

- What if you want **`stderr`** to get printed to that file as well?

```
./program > output.txt 2>&1
```

- You can also redirect only **`stderr`** to a file

```
./program 2> errors.log
```

dup() and dup2()

- If you want a new file descriptor number that refers to an open file descriptor, you can use the **dup()** function

```
int fd = dup(1); // Makes a copy of stdout
```

- It's often useful to change an existing file descriptor to refer to another stream, which you can do with **dup2()**

```
dup2(1, 2);
// Makes 2 (stderr) a copy of 1 (stdout)
```

- Now all writes to **stderr** will go to **stdout**

I/O buffering in files

- Reading from and writing to files on a hard drive is expensive
- These operations are buffered so that one big read or write happens instead of lots of little ones
  - If another program is reading from a file you've written to, it reads from the buffer, not the old file
- Even so, it is more efficient for your code to write larger amounts of data in one pass
  - Each system call has overhead

Buffering in `stdio`

- To avoid having too many system calls, **`stdio`** uses this second kind of buffering
  - This is an advantage of **`stdio`** functions rather than using low-level **`read()`** and **`write()`** directly
- The default buffer size is 8192 bytes
- The **`setvbuf()`**, **`setbuf()`**, and **`setbuffer()`** functions let you specify your own buffer

Flushing a buffer

- **`stdio`** output buffers are generally flushed (sent to the system) when they hit a newline (**`'\n'`**) or get full
  - When debugging code that can crash, make sure you put a newline in your **`printf()`**, otherwise you might not see the output before the crash
- There is an **`fflush()`** function that can flush **`stdio`** buffers

```
fflush(stdout); // Flushes stdout
// Could be any FILE*
fflush(NULL); // Flushes all buffers
```

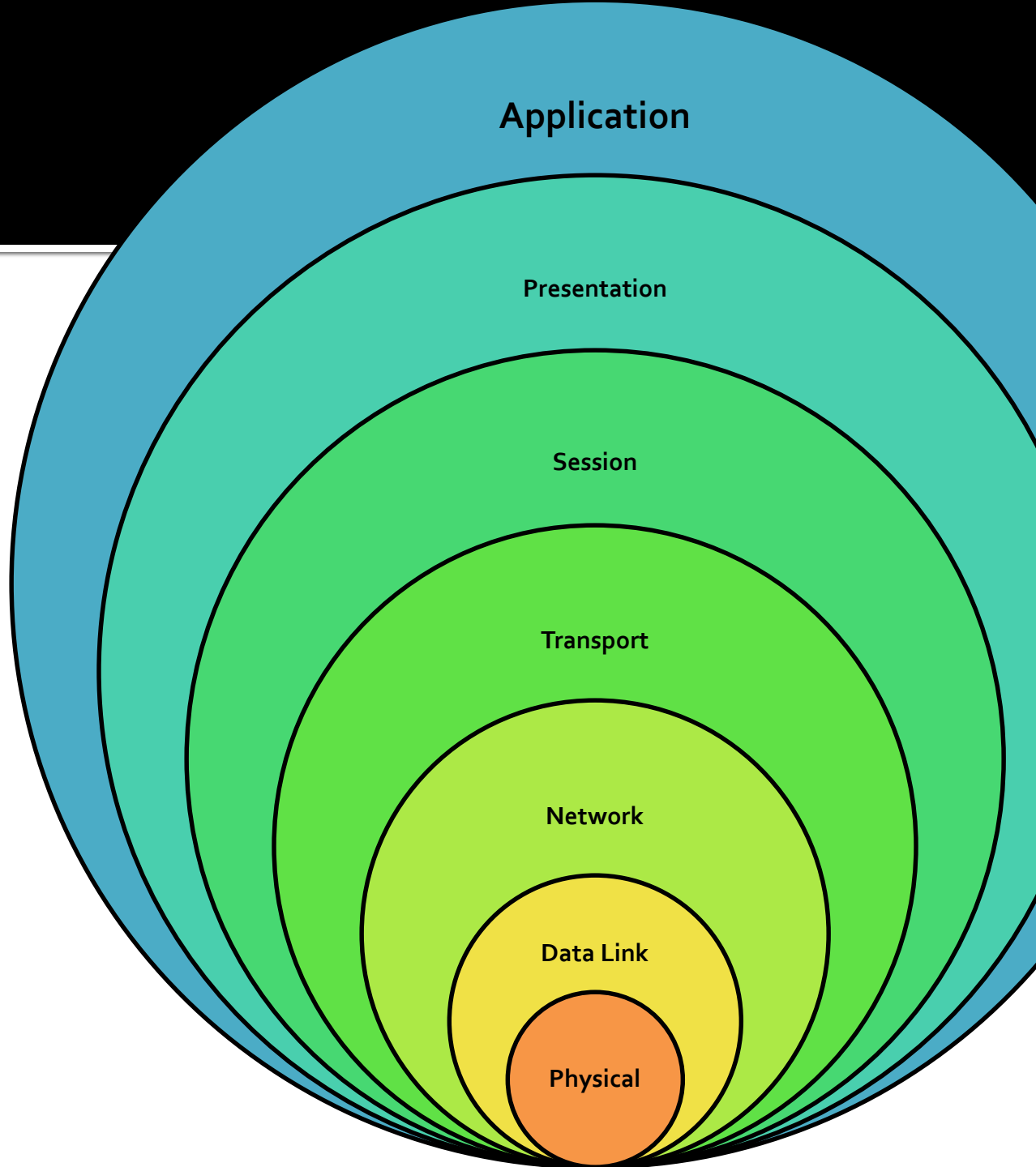
Networking

OSI seven layer model

- You can build layers of I/O on top of other layers
  - **`printf()`** is built on top of low level **`write()`** call
- The standard networking model is called the Open Systems Interconnection Reference Model
  - Also called the OSI model
  - Or the 7 layer model

Protocols

- There are many different communication protocols
- The OSI reference model is an idealized model of how different parts of communication can be abstracted into 7 layers
- Imagine that each layer is talking to another parallel layer called a **peer** on another computer
- Only the physical layer is a real connection between the two



Layers

- Not every layer is always used
- Sometimes user errors are referred to as Layer 8 problems

Layer	Name	Mnemonic	Activity	Example
7	**Application**	Away	User-level data	HTTP
6	**Presentation**	Pretzels	Data appearance, some encryption	Unicode
5	**Session**	Salty	Sessions, sequencing, recovery	TLS
4	**Transport**	Throw	Flow control, end-to-end error detection	TCP
3	**Network**	Not	Routing, blocking into packets	IP
2	**Data Link**	Dare	Data delivery, packets into frames, transmission error recovery	Ethernet
1	**Physical**	Programmers	Physical communication, bit transmission	Electrons in copper

Physical layer

- There is where the rubber meets the road
- The actual protocols for exchanging bits as electronic signals happen at the physical layer
- At this level are things like RJ45 jacks and rules for interpreting voltages sent over copper
  - Or light pulses over fiber

Data link layer

- Ethernet is the most widely used example of the data layer
- Machines at this layer are identified by a 48-bit Media Access Control (MAC) address
- The Address Resolution Protocol (ARP) can be used for one machine to ask another for its MAC address
  - Try the **`arptables`** command in Linux
- Some routers allow a MAC address to be spoofed, but MAC addresses are intended to be unique and unchanging for a particular piece of hardware

Network layer

- The most common network layer protocol is Internet Protocol (IP)
- Each computer connected to the Internet should have a unique IP address
  - IPv4 is 32 bits written as four numbers from 0 – 255, separated by dots
  - IPv6 is 128 bits written as 8 groups of 4 hexadecimal digits
- We can use **`traceroute`** to see the path of hosts leading to some IP address

Transport layer

- There are two popular possibilities for the transport layer
- Transmission Control Protocol (TCP) provides reliability
  - Sequence numbers for out of order packets
  - Retransmission for packets that never arrive
- User Datagram Protocol (UDP) is simpler
  - Packets can arrive out of order or never show up
  - Many online games use UDP because speed is more important

Session layer

- This layer isn't a key part of the TCP/IP model
- The secure sessions provided by TLS can be considered the session layer

Presentation layer

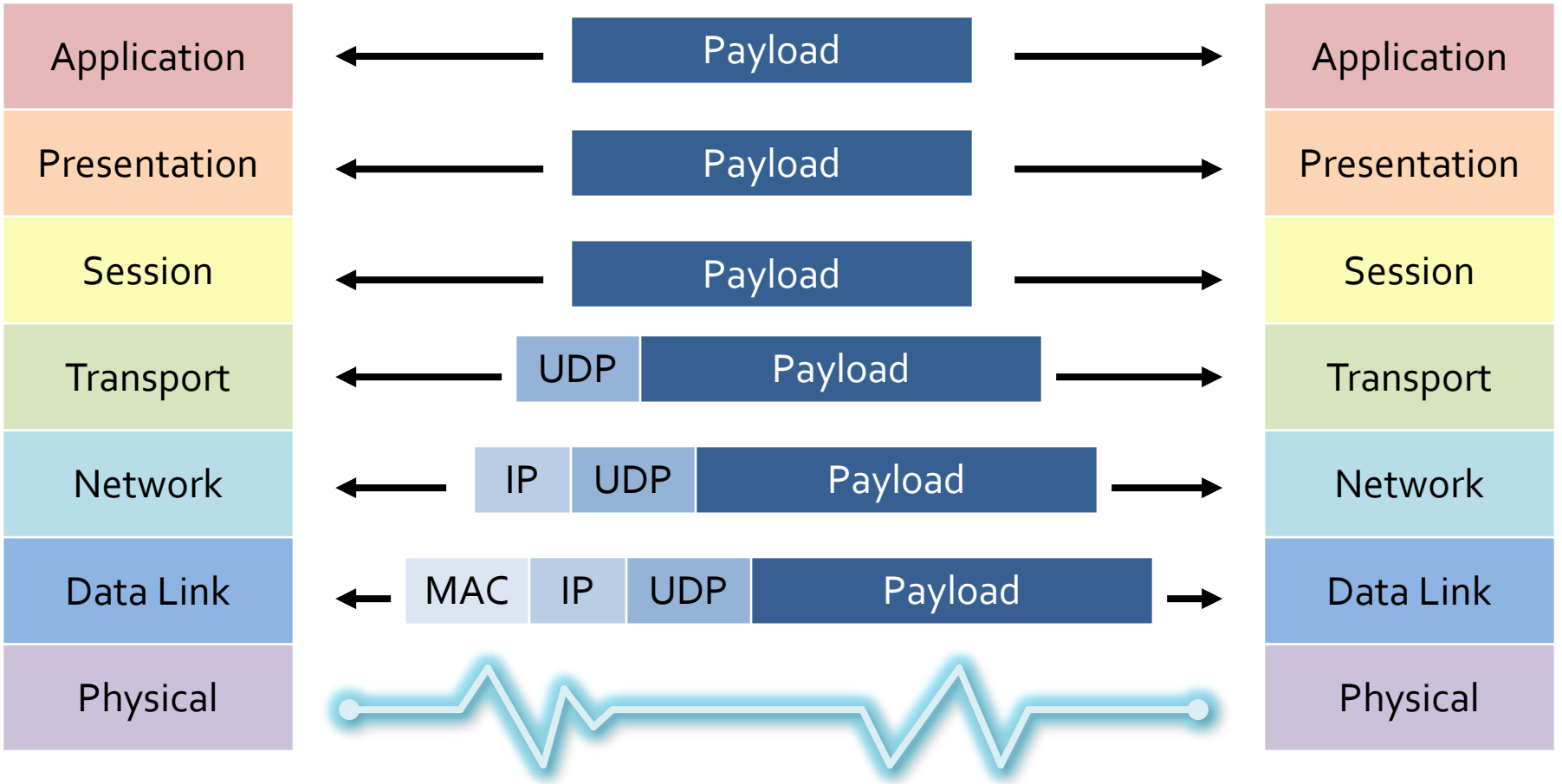
- The presentation layer is often optional
- It specifies how the data should appear
- This layer is responsible for character encoding (ASCII, UTF-8, etc.)
- MIME types are sometimes considered presentation layer issues
- Encryption and decryption can happen here

Application layer

- This is where the data is interpreted and used
- HTTP is an example of an application layer protocol
- A web browser takes the information delivered via HTTP and renders it
- Code you write deals a great deal with the application layer

Transparency

- The goal of the OSI model is to make lower layers transparent to upper ones



Mnemonics

- Seven layers is a lot to remember
- Mnemonics have been developed to help

Application	All	All	A	Away
Presentation	Pros	People	Powered-Down	Pretzels
Session	Search	Seem	System	Salty
Transport	Top	To	Transmits	Throw
Network	Notch	Need	No	Not
Data Link	Donut	Data	Data	Dare
Physical	Places	Processing	Packets	Programmers

TCP/IP

- The OSI model is sort of a sham
  - It was invented after the Internet was already in use
  - You don't need all layers
  - Some people think this categorization is not useful
- Most network communication uses TCP/IP
- We can view TCP/IP as five layers:

Layer	Action	Responsibilities	Protocols
Application	Prepare messages	User interaction	HTTP, FTP, etc.
Transport	Convert messages to segments	Sequencing, reliability, error correction	TCP or UDP
Internet	Convert segments to packets	Flow control, routing	IP
Link	Convert packets to frames	Point-to-point communication between devices on the same network	Ethernet, Wi-Fi
Physical	Transmit frames as bits	Data communication	

Upcoming

Next time…

- More networking
- Sockets

Reminders

- Work on Project 5
- Keep reading LPI chapters 13, 14, and 15