Week 1 - Wednesday

COMP 3400

Last time

- What did we talk about last time?
- Course overview
- Policies
- Schedule
- GNU style
- Systems

Questions?

Assignment 1

Fixed Width Types

The problem

- You might recall that the sizes of integer types in C are a little bit mushy:
  - `short`: at least 2 bytes
  - `long`: at least 4 bytes
  - `int`: between the size of `short` and `long`
- This can be frustrating when you need a type to hold a specific amount of data
- In Java, the sizes of types are fully specified
  - `short`: 2 bytes
  - `long`: 8 bytes
  - `int`: 4 bytes

Fixed width types

- Although it's a bit ugly, C99 specifies types with fixed sizes
- To use them, **`#include <stdint.h>`**
- Then, you're guaranteed the following:
  - **`int8_t`** 1 byte (8 bits), signed
  - **`int16_t`** 2 bytes (16 bits), signed
  - **`int32_t`** 4 bytes (32 bits), signed
  - **`uint8_t`** 1 byte (8 bits), unsigned
  - **`uint16_t`** 2 bytes (16 bits), unsigned
  - **`uint32_t`** 4 bytes (32 bits), unsigned
- And you probably get **`int64_t`** and **`uint64_t`** as well

What about printing those things?

- If you want to print an **`int`**, you use **`%d`**
- If you want to print an **`int32_t`**, what do you do?
- There are some (ugly) macros used:
  - **`PRId8`**
  - **`PRId16`**
  - **`PRId32`**
  - **`PRId64`**
- You can use these macros for octal or hex by changing **`d`** to **`o`** or **`x`**, e.g. **`PRIx32`**

Using the print macros

- To use these macros, **`#include <inttypes.h>`**
  - Note that **`inttypes.h`** includes **`stdint.h`**, so you can kill two birds with one stone
- These macros are special strings
- There's an obscure rule in C that treats consecutive strings literals like a single string literal:
  - **`"goats" "boats" "moats"`** is the same to the compiler as **`"goatsboatsmoats"`**
- To use a macro, it has to "float" in between the rest of a formatting string

```
int a = 7;
int32_t b = 7;
printf ("Value: %d\n", a);              // int version
printf ("Value: %" PRId32 "\n", b); // int32_t version
```

Course Themes

Themes from the book

- Computer systems create a platform for applications
  - They provide tools and environments that are needed for other things
  - They don't exist in isolation
- Computer systems have semiotics
  - They use symbols for communication
- Systems entail complexity
  - As they get complex, there are unintended consequences
- It's worth thinking about these themes when designing or using computer systems

Scarcity of resources

- An issue that comes up frequently when designing computer systems is scarcity of resources:
  - Computers have limited numbers of cores
  - Applications have access to a finite amount of memory
  - Networking bandwidth is limited
  - Access to shared resources has to be controlled to prevent applications from stopping each other's work or corrupting it
- The problem of scarcity can be approached, in part, with **tradeoffs**
  - Using a greater amount of one resource in order to free up another

Tradeoffs

- Space/time tradeoff
  - Sometimes using more resources can allow faster execution
  - Example: Buffer sizes in communication
  - Example: Hash tables from data structures
- Interface abstraction
  - Treating different things through a common abstraction makes a system simpler
    - But it also prevents optimization
  - Example: Linux treats networking, files, and many memory accesses like reading and writing to files
- Security vs. usability
  - Greater security always entails less usability
  - Different products need the balance at different levels

Complexity

- Complexity refers to systems with **emergent properties**
- Emergent properties are those that aren't obvious when looking at a design
- The interactions inside a system can lead to unexpected situations
- Examples:
  - There are situations where increasing the amount of system memory worsens performance
  - Deadlock is a situation where two processes trying to get access to something make it impossible for either one to get it
  - Priority inversion is a case where a high-priority process can be prevented from running by a low-priority process

Consequences of complexity

- Because of complexity, reliability is elusive
- Redundancy helps provide reliability
  - Having a back-up component ready when another component fails reduces the chance of total failure
  - Sending redundant messages can make networking protocols more reliable
- Complexity also creates misunderstanding
  - It's hard to know what really causes an error
  - Anyone who's debugged code knows this problem

Semiotics

- **Semiotics** means the use and interpretation of symbols
- The semiotics of a message involves:
  - **Syntax:** Rules for making valid messages
  - **Semantics:** The meaning of symbols
  - **Pragmatics:** Relationship between the message and the entity reading the message
- The semiotics of computer systems can involve communication between two machines, a machine and a human, or a machine and the outside world

Machine to machine communication

- Communication between machines is the easiest of the three to describe
- However, machines send and receive bits that mean nothing without context
- Consider the bit sequence 10101110, which could be interpreted in C as:
  - Signed integer: **`-82`**
  - Unsigned integer: **`174`**
  - Character: **`'®'`**
  - Bitmask: **`0xAE`**

Machine to human communication

- Humans don't understand computers at an instinctive level
- CS education is a kind of brain damage to make us think more like machines
- If you don't quite understand the semantics of a line of C or Java, you might be surprised by the output
- But other things like the concept of "happening at the same time" mean different things to humans and computers
- Example:
  - `int x = 14;`
  - Now, these lines of code execute on two different threads, at *nearly* the same time:
  - `x = x + 1;`
  - `x = x - 1;`
  - What are the possible values of **`x`**?

Machine to world communication

- Computers are models of the real world
- We mostly ignore that and treat the computer as a kind of reality
- Problems that crop up:
  - Floating-point values are only approximations of real numbers
    - Even "simple" numbers like 0.1 can't be exactly represented with the usual system
  - The values 0 and 1 are represented by voltages
    - Occasionally, (especially when overclocking), a voltage that should be a 1 is read as a 0, or vice versa
  - Cosmic rays occasionally flip bits inside our machines
  - Getting machines to agree on issues of timing is difficult

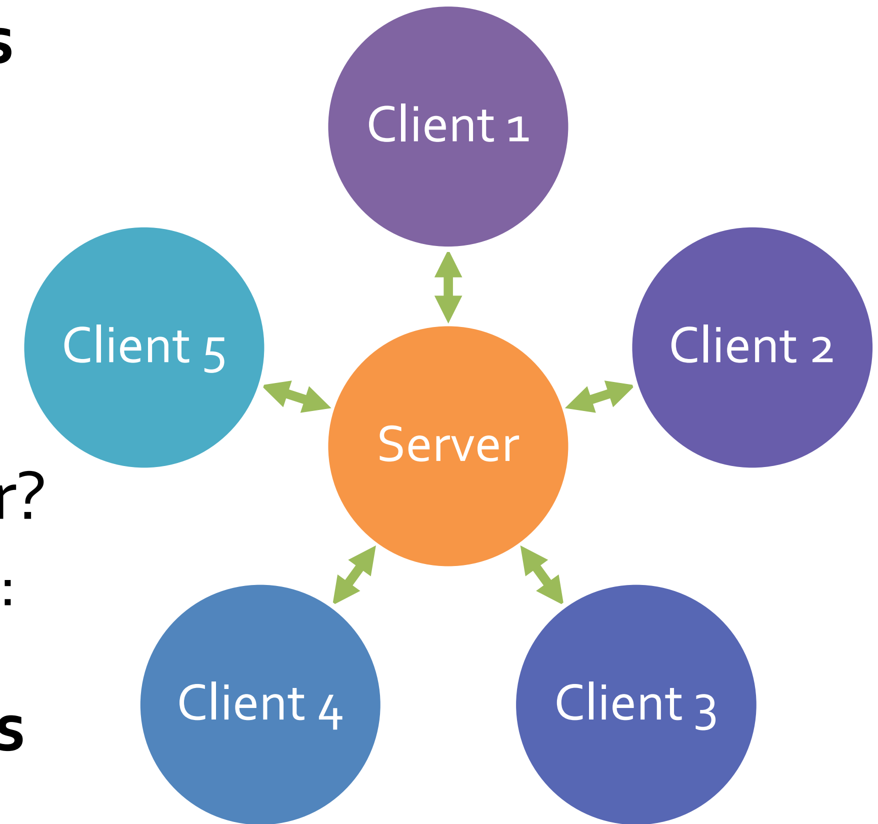
System Architectures

System architectures

- System architectures are models of systems that describe:
  - Relationships between entities in the system
  - Ways the entities communicate
- Different architectural styles have pros and cons
- Using a certain style can have big impacts on system performance
- Common styles:
  - Client/server
  - Peer-to-peer (P2P)
  - Layered
  - Pipe-and-filter
  - Event-driven
  - Hybrid

Client/server architectures

- This book considers **client/server architectures** from the perspective of a many clients connecting to a single server
  - If you recall, the Software Engineering book describes client/server as a system with many servers, each of which offer a single service
- How does a client know how to reach the server?
  - Uniform resource identifier (URI) is a common way: www.goats.net/image.jpg
- Client/server architectures depend on **protocols** to define how clients can request services and understand the response



Client/server advantages and disadvantages

ADVANTAGES

- Updates are simple, because only the server needs to be updated
- Only the server needs to be checked for security problems or data corruption

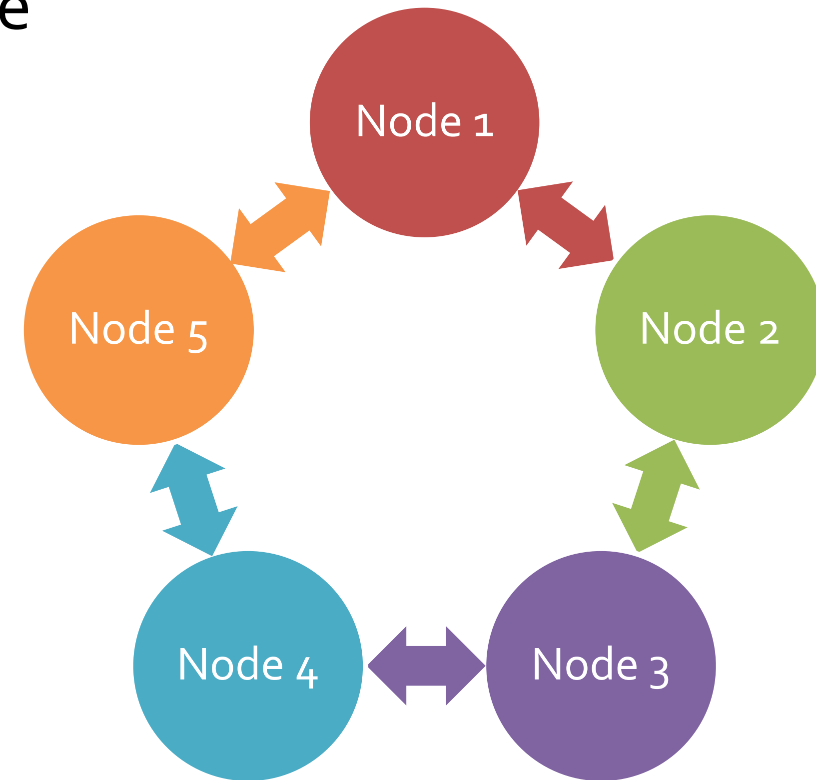
DISADVANTAGES

- Single point of failure

- To reduce the single point of failure problem, it's common to have multiple servers that offer the same services or files
- To work, these servers must coordinate with each other when one is updated

Peer-to-peer (P2P) architectures

- If more and more servers are used, the architecture begins to look like a **P2P architecture**
  - BitTorrent
  - DNS
- In P2P, there is usually no distinction between clients and servers, since most entities act as both
- Advantages:
  - Service scales, staying the same or improving as the number of users goes up
- Disadvantages:
  - Security: A corrupted node can be hard to detect
  - Administration: Propagating changes can be difficult



Layered architectures

- **Layered architectures** divide systems into a strict hierarchy of components
- Each layer can only communicate with the layer above and below it
- Advantages:
  - As long as a new layer knows how to talk to the layer above and below, it can be swapped out with an old layer
  - New layers can be added on top
- Disadvantages:
  - It's hard to divide systems into hierarchical layers
  - It can be inefficient to prevent one layer from talking directly to one much lower or higher
  - Some services at each layer are redundant

Presentation Layer

Business Layer

Services Layer

Persistence Layer

Pipe-and-filter architectures

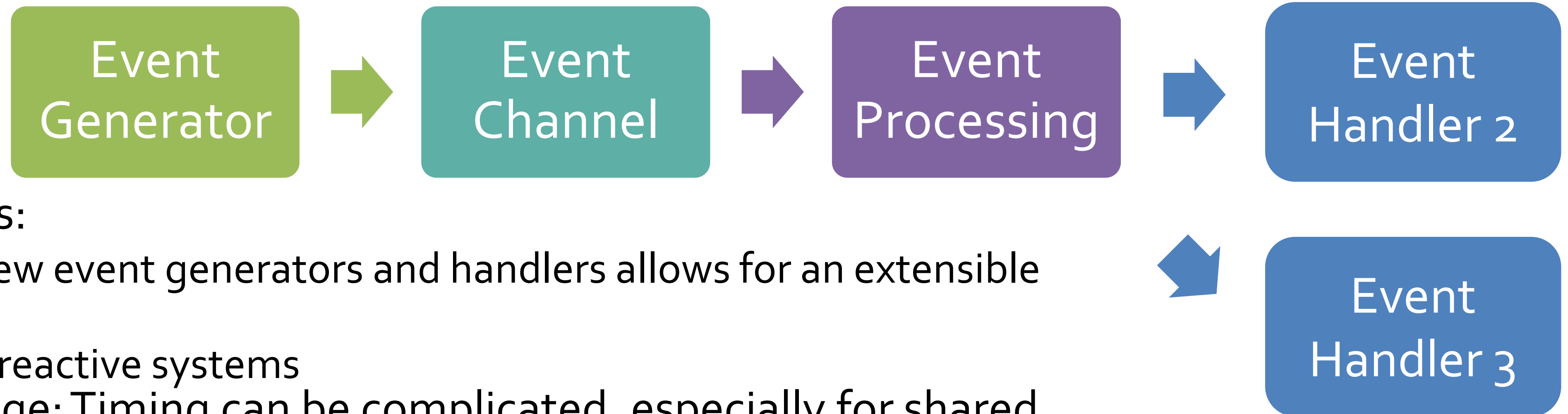
- **Pipe-and-filter architectures** send data in one direction through a series of components
- The output of one stage is the input of the next
- Each stage transforms the data in some way
- Examples:
  - Linux command-line piping

```
sort foo.txt | grep -i error | head -n 10 > out.txt
```

  - Java stream filtering
  - Stages of a compiler
- Advantages:
  - Good for serial data processing
  - Modular components that have the same input and output can be reused in different sequences
- Disadvantage: No error recovery if something breaks in the middle

Event-driven architectures

- **Event-driven architectures** react to events, changes in the state of the system
  - GUIs are a common example of event-driven architectures
- Event generator create events
- Event channels send the event to the appropriate event handlers



- Advantages:
  - Adding new event generators and handlers allows for an extensible system
  - Good for reactive systems
- Disadvantage: Timing can be complicated, especially for shared resources

Hybrid architectures

- We talk about the previous architectures because they're models that have been successful in the past
- Most real systems are a mix of different architectures
  - The whole system could be one architecture, but its components have their own
  - A system could be mostly one architecture but break a couple of rules
  - There can be different ways of looking at the same system
- Example: OS kernel
  - Event-driven because it has interrupt handlers to respond to signals from the hardware
  - Client/server because applications that make system calls are making requests
  - Layered because file systems and networking operate with layers from the generic operation down to the requirements of particular hardware

Upcoming

Next time…

- State machines
- UML

Reminders

- **Form teams for Assignments 1 and 2 and Project 1 if you haven't**
  - Assignment 1 is due **next** Friday!
- Read sections 1.5 and 1.6