EC 535: Introduction to Embedded Systems

Eshed Ohn-Bar

1/22/2025

TA Office Hours:

4-5PM, Tu/Th, PHO 307

Homework

In-Class Exercises

Challenges in embedded system design

- How much hardware do we need?
  - How many processors? How big are they? How much memory?
- How do we meet performance requirements?
  - What's in hardware? What's in software?
  - Faster hardware or cleverer software?
- How do we minimize cost and power?
  - Turn off unnecessary logic? Reduce memory accesses?
- How do we ship in time?
  - Buy solution? From scratch? Off-the-shelf chips? IP-reuse?

Optimization by doing things in a certain order

constraint to make system responsive in real-time, need to do optimization on SW and/or HW

→ mechanisms to preserve energy

↳ different ways to speed up process is outsourcing parts

automotive industry

AI solutions bottlenecked into cuda and NVidia

optimizing codes req. optimize CUDA code → very hard...?

Challenges (cont'd)

testing ABS is hard to do
- Simulating hardware

- How do we know that it fully works? extensive testing
  - Is the specification correct?
  - Does the implementation meet the spec?
  - How do we test for real-time characteristics?
  - How do we test on real data?

specifications to verify

functional aspects to verify and see if we meet them in testing

- How do we locate the problem if it doesn't work?
  - Observability, controllability?

- hard to debug embedded systems because no monitoring, various debugging tools but more difficult than general purpose systems

difficult to do real-world testing for embedded systems

☆ HW in-the-loop simulation

↳ phys controller testing in loop
- simulated inputs to system in closed environment

Embedded System Designers

- Expertise with both **software and hardware** is needed to optimize design metrics
  - Not just a hardware or software expert.
  - A designer must be comfortable with various technologies in order to choose the best for a given application and constraints.
  - A designer must be able to communicate with teammates of various backgrounds.

domain knowledge

need to approach in all aspects

Embedded Systems – Design methodologies

- A procedure for designing a system.
- Understanding your methodology helps you ensure you didn't skip anything.
- Compilers, software engineering tools, computer-aided design (CAD) tools, etc., can be used to:
  - help automate methodology steps;
  - keep track of the methodology itself.

automate parts of the design process
Where it can check system automatically.
★Track design process

↳ Git!! Version control

Top-down vs. bottom-up

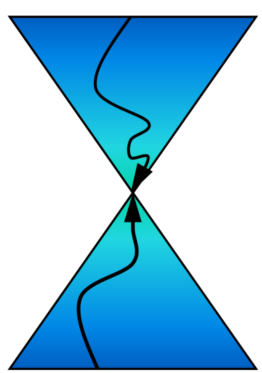
- Top-down design:
  - start from most abstract description;
  - work to most detailed.
- Bottom-up design:
  - work from small components to big system.
- Real design uses both techniques, more or less.

What the system needs to do

"I want a car to get me to the destination"

↳ think about what customer wants. often want to do things but we dont have avail. resources

→ figure out what we have integrate and build up to what we can achieve



application

architecture

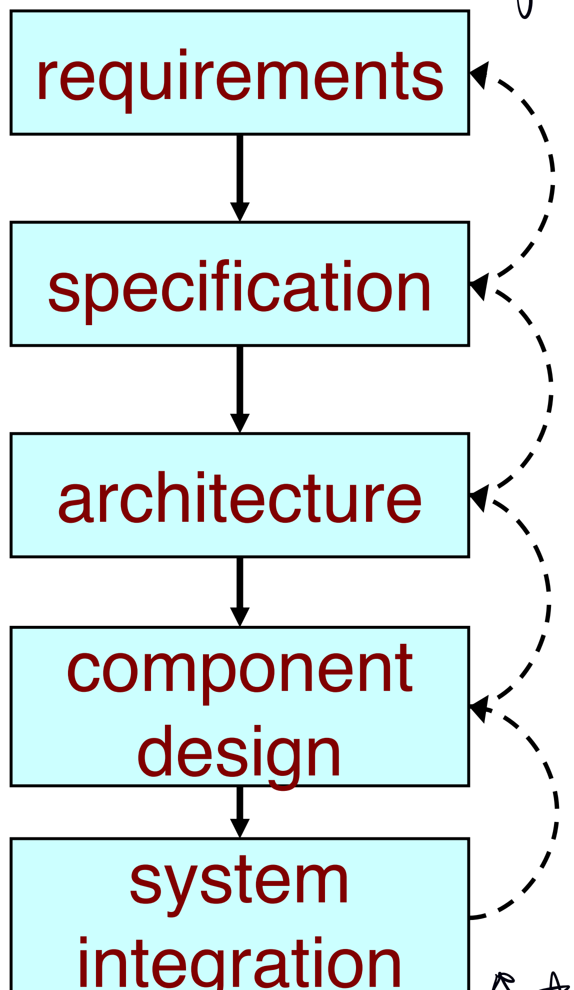
component

☆ combine approaches

☆ apple display: design from scratch

Levels of abstraction

high level



↖ ☆ overlooked aspect of engineering
lots of things can go wrong

Requirements

expectations in language for system

- Plain language description of what the user wants and expects to get.
- May be developed in several ways:
  - talking directly to customers;
  - talking to marketing representatives;
  - providing prototypes to users for comment.
- Functional requirements:
  - output as a function of input.
- Non-functional requirements:
  - time required to compute output;
  - size, weight, etc.;
  - power consumption;
  - reliability;
  - etc.

*What types of requirements require less redesigns

Specification

- A more precise description of the system:
  - should not imply a particular architecture;
  - provides input to the architecture design process.

- May include functional and non-functional elements.

system behavior
input → output

performance constraints

- May be executable or may be in mathematical form for proofs.

* formal methods?

Architecture design

- What major components do we need to satisfy the specification?
- Hardware components:
  - CPUs, peripherals, etc.
- Software components:
  - Major programs and their operations.
- Must take into account functional and non-functional specifications.

Designing hardware and software components

algorithm to adjust temp
software-based

- Must spend time architecting the system before you start coding.

- Some components are ready-made, some can be modified from existing designs, others must be designed from scratch.

SW – flexible part of system
HW – inflexible but great rewards for optimized

GPU: market share dominated by Nvidia,
don't need to modify too much

Some bought unchanged,
power management algo is new,
modified code

} → hard task to do new products

System integration

undervalued
hard to do

- Put together the components.
  - Many bugs appear only at this stage.

- Have a plan for integrating components to uncover bugs quickly, test as much functionality as early as possible.

Assumptions on HW and SW

everything synchro to clock

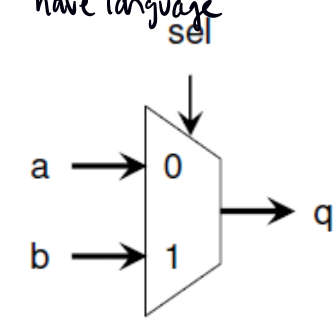
- **Hardware:** In this course, *hardware* means *single-clock synchronous digital circuits* that are created using word-level combinational and sequential building blocks. These circuits can be modeled with building blocks such as registers, adders, multiplexers, … perform various ops

  basic building blocks of embedded system

  - There are well-known circuit symbols for these building blocks.
  - Using these circuit elements, schematics can be created.
  - Schematics quickly become incomprehensible with increasing circuit complexity, hardware designers very often use a textual representation in practice.

q = if (sel) then ~~a~~ b else ~~b~~ a;

diagrams can get pretty messy, so might have language



Other Types of HW?

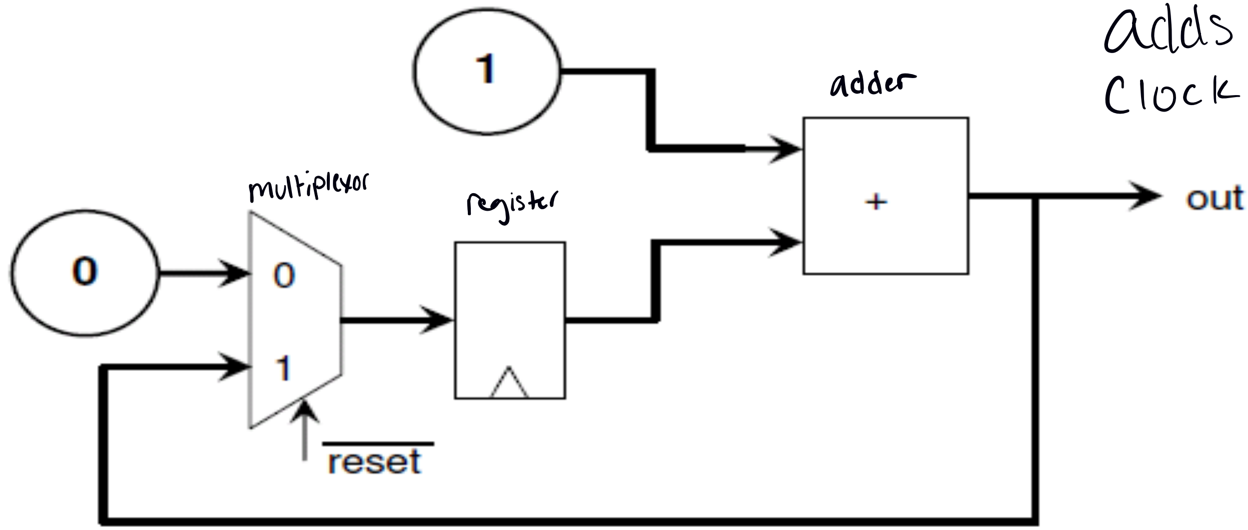
★ course is synchronous single-clock
easier to work with

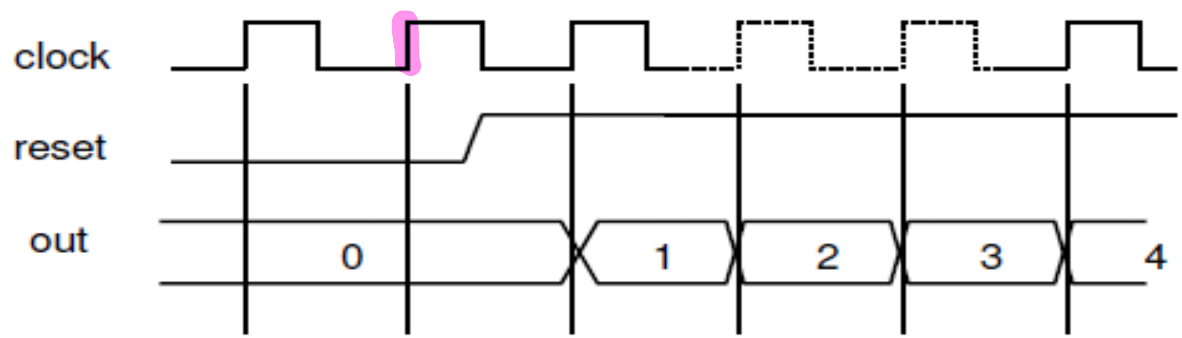
- Asynchronous hardware
- Dynamic logic
- Multi-phase clocked hardware
- etc.

→ We'll focus on synchronous single-clock hardware

Synchronous HW Model

adds 1 every time
clock edge hits





cycle	1	2	3	4	5	6
reset	0	0	1	1	1	1
out	0	0	1	2	3	4

Software

One processor one task for real-time components
One op. at any time : issue

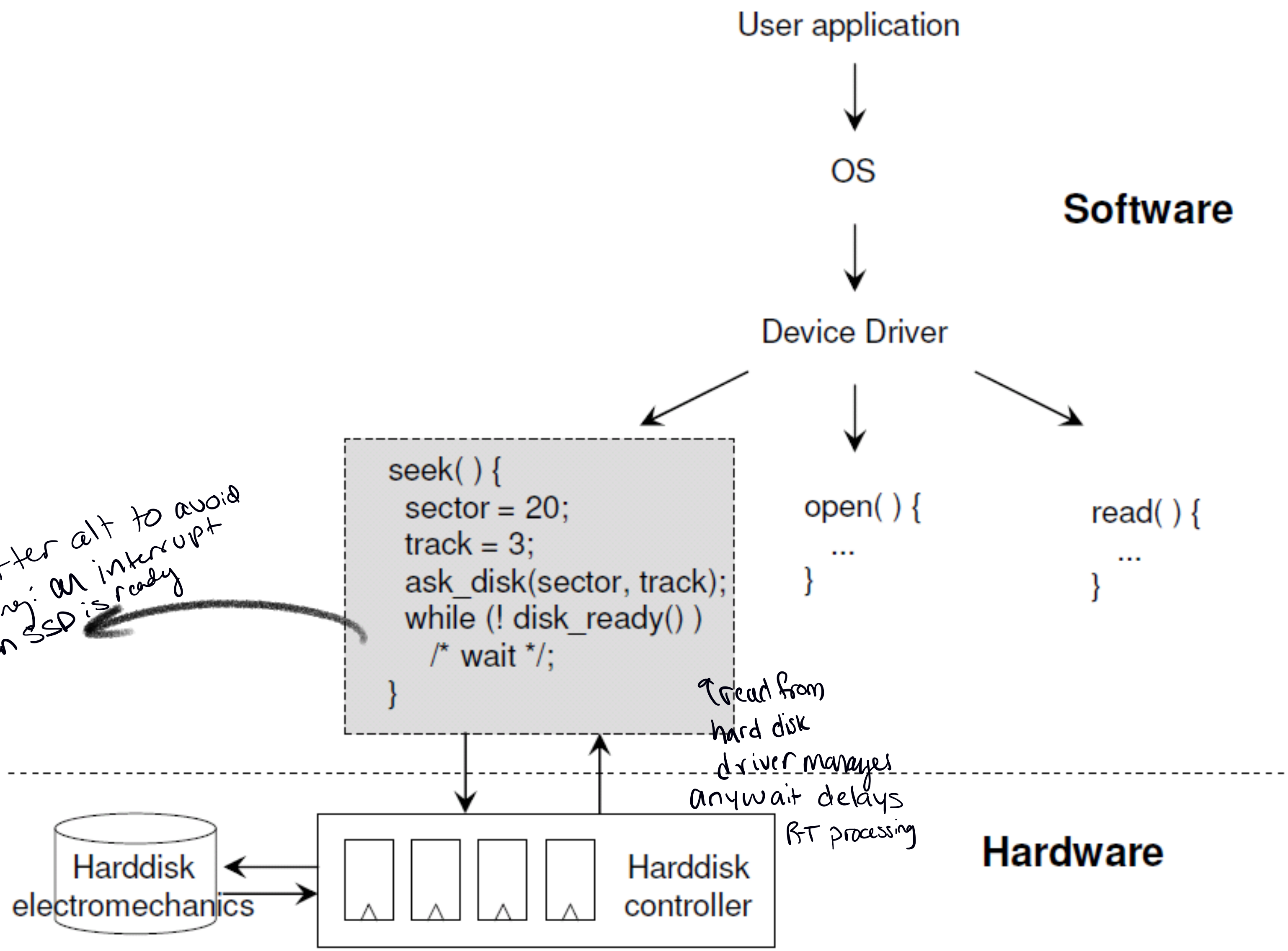
- With *software* we [mostly] mean *single-thread sequential programs,* such as those represented by a C program or an assembly program.

- We make abstractions of underlying mechanisms to enhance concurrency (such as an operating system).

allows multiple things
to 'run' at same time

Device Driver for Hard Disk

interface between SW + HW



better alt to avoid waiting: an interrupt when SSD is ready

read from hard disk
driver manages
anywait delays
RT processing

Comparing Hardware and Software

HW w/ time and clock

HW SW

- Parallel versus sequential operation
- Spatial versus temporal (time-wise) decomposition
- Flexibility
- Control processing versus data processing
- Modeling and implementation
- Intellectual property reuse

Main takeaway

Concurrency and Parallelism

OS can manage multiple tasks concurrently

- Concurrency is the ability to execute simultaneous operations (because these operations are completely independent).
  - Relates to an application model.

SW sequential but can incorporate concurrency

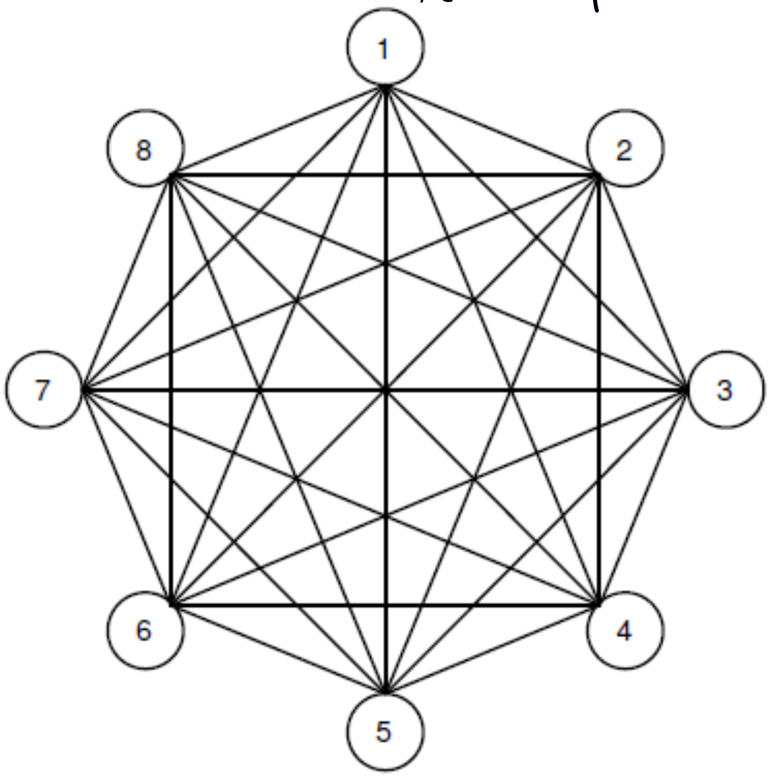
- Parallelism is the ability to execute simultaneous operations because the operations can run on different processors or circuit elements.
  - Relates to the implementation of the application model.

HW usually parallel

- HW: Parallel?
- SW: Sequential, concurrent, parallel?

An example parallel processor: Connection Machine (CM)

network of nodes, running own C; can be paralelized



Danny Hillis, 1980s, MIT

- How to write programs for such a parallel architecture?
  - Individual C programs for each node?
    - Complexity
    - Scalability
  - Concurrent algorithms

CUDA :. some easier to parallelize

GPU might be slower than CPU (all unique)

but if component can be blocked and identical, GPU faster

Conclusions about concurrency

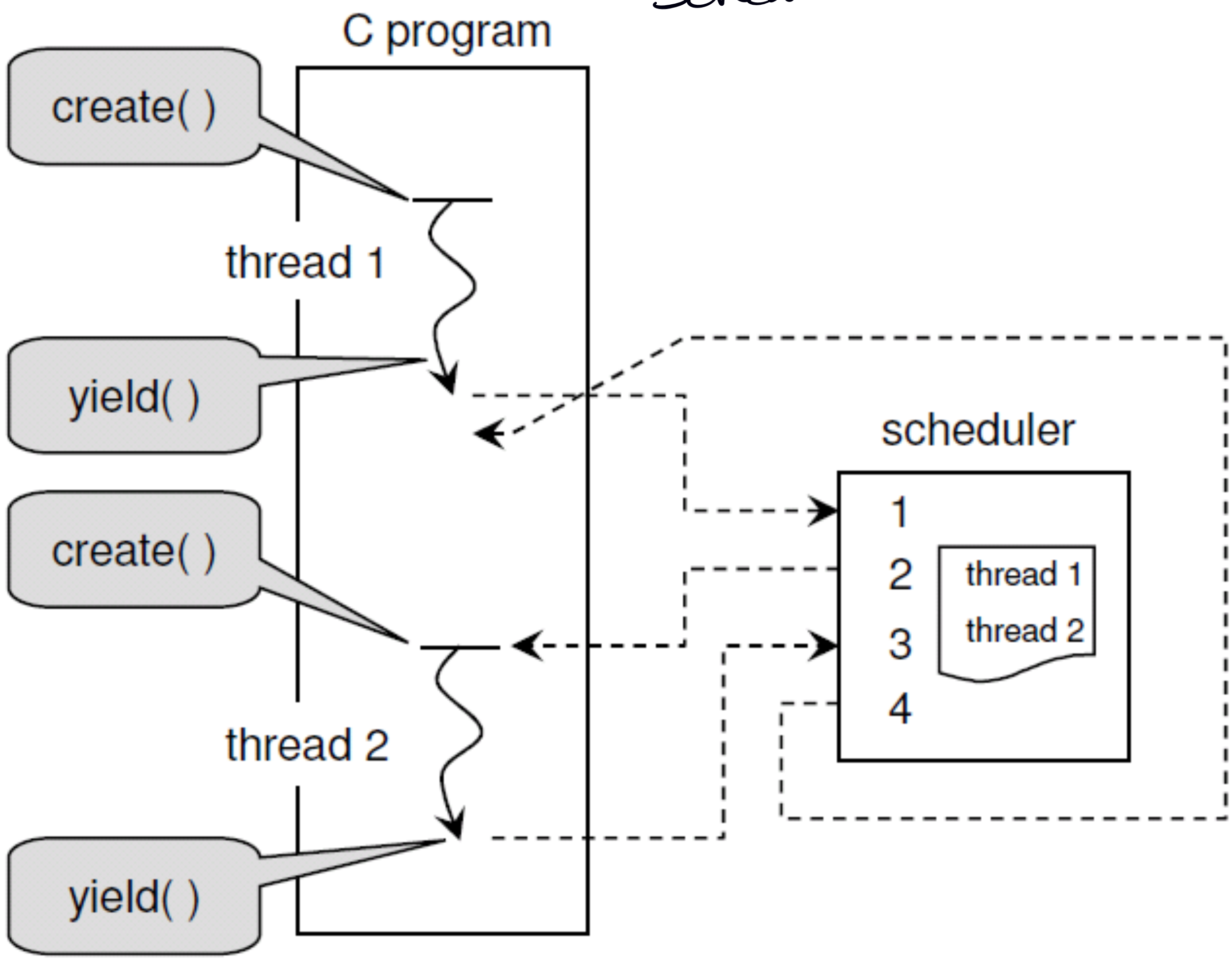
- If you develop a concurrent specification, you will be able to make optimal use of the underlying parallel hardware.
- In contrast, if you restrict yourself to a sequential specification from the start, it will be much harder to make use of underlying parallel hardware.
- Do not settle for a sequential language (such as C) as a universal specification mechanism. The use of C is excellent to make an executable, functional model of what you want to build.
  - But there exist concurrent specification mechanisms *(we will discuss later)* that may be better suited for parallel implementation than C.

Cooperative Multithreading

multithreading
two separate tasks, yield allows
schedule to switch to other operation

developer
inserts

∴ issue with
inefficiencies



Why do we need a scheduler?

based on yield or other things ∴ other policies

- Because we want to make sure that each thread gets a chance to run. The threads themselves do not have to know what other threads are active in the system.
- Because we want to have *yield points* at any point in a C program, even in the middle of a function. C does not allow to 'suspend' a function and then later resume it.
- Scheduling policies:
  - Round-Robin
  - Priority
  - More details coming up later in the course.

Cooperative Multithreading Example

- An example cooperative multithreading library: *quickthreads*.
- quickthreads API consists of four function calls:
  - stp_init( ) initializes the theading system
  - stp_create(stp_userf_t *F, void *G) creates a thread that will start execution with user function F. The function will be called with a single argument G. The thread will terminate when that function completes, or when the thread aborts.
  - stp_yield( ) releases control over the thread to the scheduler.
  - stp_abort( ) terminates a thread, so that it will be no more scheduled.

Sample code using *quickthreads*

```
#include "../qt/stp.h"
#include <stdio.h>
void hello(void *null) {
  int n = 3;
  while (n-- > 0) {
      printf("hello\n");
      stp_yield();
  }
}
void world(void *null) {
  int n = 5;
  while (n-- > 0) {
      printf("world\n");
      stp_yield();
  }
}
```

```
int main(int argc, char
  **argv) {
  stp_init();
  stp_create(hello, 0);
  stp_create(world, 0);
  stp_start();
  return 0;
}
```

Summary

- With hardware, we will refer to **single-clock synchronous digital circuits**, and with software we will refer to **single-thread sequential programs**.
- Hardware and software have complementary properties for many different aspects, and are in fact each others' dual in many respects.
- The opposite of concurrency is **sequential**. Concurrent specifications are preferable over sequential ones, because it is easier to convert a concurrent specification to a parallel one than it is to parallelize a sequential specification.
- There are several modeling abstraction levels for software and hardware: **transaction-level, instruction-accurate, cycle-accurate, event-driven and continuous-time.**

Reading

For those interested (optional):

- Hillis and Steele, "Data Parallel Algorithms", 1986:

  http://cva.stanford.edu/classes/cs99s/papers/hillis-steele-data-parallel-algorithms.pdf

- David Patterson, "The Trouble with Multicore", 2010:

  http://spectrum.ieee.org/computing/software/the-trouble-with-multicore

Everyone:

For next lecture:

- If you have not taken a computer organization/architecture course before:
  - Study the basic **5-stage RISC processor pipeline**
    - http://en.wikipedia.org/wiki/Instruction_pipeline
    - http://en.wikipedia.org/wiki/Classic_RISC_pipeline
    - Hennessy & Patterson Computer Organization / Computer Architecture books are great resources

In Class Exercise (getting started for HW1)

A simple *pthreads* example

```
#include <stdlib.h>
#include <pthread.h>

 void *myFirstThread(void *p)
 {
    sleep(1);
    printf("I am the first thread \n");
    return NULL;
}
void *mySecondThread(void *p)
{
     printf("I am the second thread \n");
     pthread_yield();
}
void *myNewThread(void *p)
{
        printf("Hello!\n");
}
```

In Class Exercise (getting started for HW1)

```
int main()
{
        pthread_t tid,tid2, tid3;

        printf("Program started...\n");

        pthread_create(&tid, NULL, myFirstThread, NULL);
        pthread_create(&tid2, NULL, mySecondThread, NULL);
        pthread_create(&tid3, NULL, myNewThread, NULL);

        pthread_join(tid, NULL);
        pthread_join(tid2, NULL);
        pthread_join(tid3, NULL);

        printf("All threads done.\n");
        exit(0);
}
```

- What is the output?
- What happens if you do not use “join”?

Compile with: gcc hw1intro.c -lpthread -o hw1intro

```c
#include <stdlib.h>
#include <pthread.h>

 void *myFirstThread(void *p)
 {
    sleep(1);
    printf("I am the first thread \n");
    return NULL;
}
void *mySecondThread(void *p)
{
     printf("I am the second thread \n");
     pthread_yield();
}
void *myNewThread(void *p)
{
        printf("Hello!\n");
        //Ask user to enter an integer
        //print that integer in decimal, binary,
       //and in hexadecimal form
}
```

in-class exercise to prepare for HW1

Email your modified in-class exercise code (hw1intro.c) to:

ece535submit@gmail.com

<u>with subject line:</u>

yourBUusername_hw1inclass