Threads

- OS abstraction for virtualizing the CPU
  - Each thread runs on CPU, thinks it has its own set of CPU registers, unaware of other threads

- # of threads is arbitrary, # of CPUs is fixed
  - Implemented by interleaving thread instruction streams

Concurrent execution:  
Overlapping time intervals

Parallel execution

Time, uniprocessor

Time, dual processor
Understanding Threads: Single CPU

```c
main() {
    thread_create(T1);
    thread_create(T2);
    main_loop();
}

T1() {
    t1_loop();
}

T2() {
    t2_loop();
}
```

CPU → main → T1 → T2 → T1 → main → T2 →

- `thread_create(T1)`
- `thread_create(T2)`
- `main_loop()`
- `t1_loop()`
- `t2_loop()`
- `thread_switch: suspend + resume`
Understanding Threads: Multiple CPUs

main() {
    thread_create(T1);
    thread_create(T2);
    main_loop();
}

T1() {
    t1_loop();
}

T2() {
    t2_loop();
}

CPU 1  main → T2
      thread_create(T2)
          main_loop
      thread_create(T1)

CPU 2  T1 → main → T2
      t1_loop
      thread_switch: suspend + resume

      T1 → main_loop → T2
      t2_loop
Benefits of Threads

- Allow running multiple programs, or multiple tasks in a program 1) concurrently, or 2) in parallel

  - Concurrently: Run threads on same CPU
    - Helps hide I/O latency: if one thread has to wait on I/O, then another thread can use the CPU while the first thread blocks
      - Enable using CPU and devices in parallel
    - E.g., run interactive programs like web browser, text editor

  - Parallel: Run threads on different CPUs
    - Helps speedup computation
      - Enable using different CPUs in parallel
    - E.g., run video encoder, matrix multiply program
Summary

- A thread virtualizes the CPU
  - Multiple threads can run concurrently or in parallel
  - Threads are unaware that other threads are running

- Concurrent execution improves performance when programs perform many IO operations

- Parallel execution improves performance when programs need to use CPU continuously

- The OS implements threads by interleaving their execution on the CPU
  - Requires suspending and resuming threads
Think Time

- What is the difference between a thread and a function?
- The OS switches threads frequently on a CPU and so user think that all threads are running. Can you think of other ways that users are “fooled” by timing-based optical illusions in real life?
What is the difference between a thread and a function

- A thread runs one or more functions on a single stack. Suppose within a thread, function F1 calls F2. Until F2 completes, F1 will not run again. This means that even if F2 were to run on another processor, we could not run F1 and F2 in parallel. On the other hand, threads are independent streams of execution. They are unaware of each other and run independently of each other. This means that a thread does not have to finish running before another thread can run. Each thread calls its own set of functions and has its own stack.
Think Time Answers

- The OS switches threads frequently on a CPU and so user think that all threads are running. Can you think of other ways that users are “fooled” by timing-based optical illusions in real life?
  - A real world analogy of an optical illusion is a wheel that is spinning fast -- it may appear to spin in the opposite direction: https://www.youtube.com/watch?v=Sm10zvZ7fvg