Thread Pools

A Useful Tool for Easy Concurrency

CSE333 Section 9 (5/26/11)

Colin Gordon (csgordon@cs)

What's a Thread Pool?

- An abstraction for a group of threads where
  - Some number of threads are always present, sometimes idle
  - Clients drop work items into a queue, and worker threads pick them up when ready
- Advantages:
  - Don't have to manually manage threads
  - When the tasks are independent, it's safe easy parallelism
  - Usually little per-thread startup cost

Why Not Just Non-blocking IO?

- How many tasks can a single thread using select() or poll() manage concurrently?
  - Can track many
  - Can only run one at a time!
    - If select() returns more than 1 ready file descriptor, some task that could run sits idle while the first descriptor is handled

Who Uses Thread Pools?

- Everyone!
  - Microsoft
    - WIN32 ThreadPool API
    - C#/.NET System.Threading.ThreadPool
  - Apple
    - Grand Central Dispatch
  - Linux
    - Everyone writes their own
  - ~~Sun~~Oracle
    - java.util.concurrent.Executors.*

How Do I Use a Thread Pool

- Interfaces vary, but have common core functionality
  - Create a pool of *n* threads
  - Set up a work function for each thread, which takes an argument for the work to do (a *task*)
  - Whenever there is work to be done, add a task to the queue
    - Next time a thread in the pool is available, it will grab the task and call the work function

How Do I NOT Use a Thread Pool?

- For now, never have multiple tasks touch the same data structure
  - This is called a data race or race condition, and can be very hard to debug.
  - Can be done safely using mutexes, etc., which we aren't teaching here; see 451
- Don't let the pool create many more threads than you have CPUs
  - Makes the threads fight for CPU time, while having n+2 to 2*n threads for n CPUs keeps everyone busy, but not too busy depending on the tasks

Homework 4's ThreadPool

to ThreadPool.h, EventLoop.h, and EventLoopHandler.h!