Concurrency

Concurrency

- Concurrency vs parallelism?
- Concurrency: one system doing multiple things. They might be done in parallel, they might not.
- Parallelism: doing two tasks at once.
- Before multicore systems, how did multiple things get done?
- “Concurrency is not Parallelism” by Rob Pike

Concurrency and Networking

- Why do concurrency and networking often go together?
- The network is slow (compared to the computer)!
- What do you do while you wait?

Concurrency

- Four major methods:
  - Processes
  - Threads (most common)
  - Micro-threads
  - Async (does not provide parallelism)

Processes

- Pretty much the same as starting your program twice at the same time
- Whole program is copied
- Slow and heavy
- No shared memory between processes
    - Requires explicit sharing, such as a file or a socket

Threads

- Lighter alternative to processes
  - 10x faster creation than processes
- Threads provided by OS
- Memory is shared between threads (good and bad)

Threads

Shared Memory

- How do you deal with shared memory?
- What if two threads are writing to one memory location at once?
- Atomic operations
- Synchronization techniques
  - Locks/Mutex
  - Semaphore
  - Message Queue

Micro-threads

- Instead of OS controlled threads, programming language controlled threads
- Much lighter weight than normal threads
- Let the library deal with OS threads

I/O Multiplexing

- While you are waiting for I/O, do something else
- Give operating system list of sockets (or files) you are waiting for
- OS will let you know when one is ready to be acted on
- select, poll, epoll

Async

- One task runs at a time, but never blocks
- Tasks explicitly yield control to another task
- An event loop manages which tasks are ready to run
- Does not give you parallel execution
- You don’t have to worry about the problems associated with parallel execution

Python

- Python has strange behavior for concurrency
- Global Interpreter Lock
- Not a problem for these labs because you are waiting for the network

Thread/Process Pool

- It is expensive to keep creating new threads/processes for every job
- Keep a list of threads/processes
- Give a task to an available thread/process
- If no threads/processes are available, block
- Once a thread/process is done, it is given another task

Some programmers, when confronted with a problem, think “I know, I’ll solve it with threads!”. have Now problems. two they