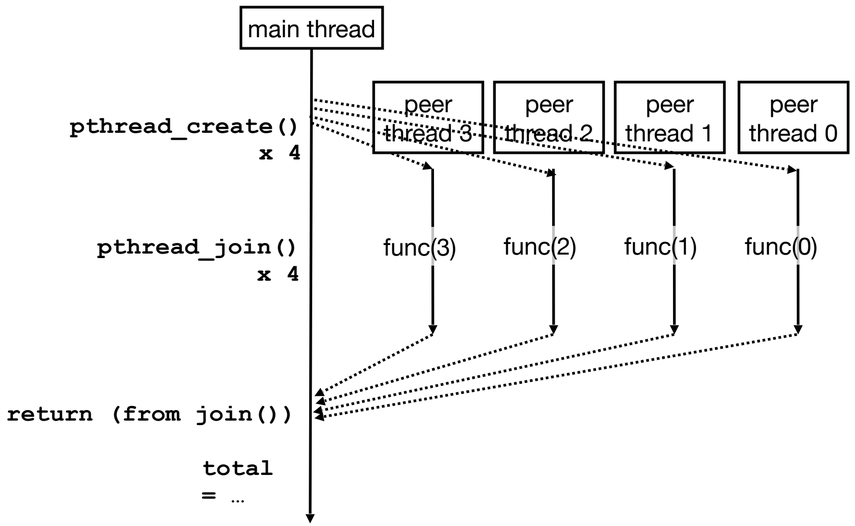
Thread Demo

CS143: lecture 17

Byron Zhong, July 25

Moral: `slowprimes.c`

- C is so much better than Python at number crunching
- If a problem can be broken down into disjoint subproblems, parallelism shortens the running time by a lot



Moral: `primes.c`

- The best kind of performance improvement is algorithmic improvement -- the one that improves the asymptotic running time.
- Thread creation has its own cost. This cost needs to be factored in when one decides whether to use concurrency.
- Multithreading loses some benefit when the problem cannot be cleanly sliced into subproblems
- Multithreading code is usually a lot more complicated than single-threaded code.

Moral: `badcnt.c`

- Concurrency bug can happen when threads share resources.
- Threads do not have its own virtual memory space like processes do
  - Memory access is not atomic -- x += 1 is three separate steps
    - Read x to a register (load)
    - Increment the register (update)
    - Write the register to memory (store)
  - Memory updates need to be synchronized

Moral: `goodcnt.c`

- Synchronization can be done via `semaphores` (e.g. locks)
  - `sem_wait` waits the value to be positive
  - `sem_post` increments the value
- Synchronized code is not parallel :(

Moral: `bomb.c`

- Asynchronous I/O
- Semaphores can be used to indicate the readiness of a value

When should I use threads?

- Spot parallelizable code
  - Code that doesn't depend on other's result
  - E.g. loop where each iteration is independent
- Beware of concurrency bugs
  - Read-write conflict
  - Write-write conflict
  - Semaphore