Skip: Thread Library

Thread Libraries: NB

Threading issues + OS not important

**Motivation**

Threads run within a process.

Process creation is **heavy** weight, while thread creation is **lightweight**.

Characteristics about a process?  
-Has a unique ID assigned by OS  
-Has an address space [starts + ends at specific address of code, + heap + stack ect]  
-Always has a parent (mostly)  
-Process has a state (waiting, running)

The way a process starts:   
Parent process creates system call.  
Then kernal does extension job eg allocating memory, resources, processID, state management (All stored in process control block PCB).

The way a thread starts:  
No need to assign address space, as already has process memory ect.  
Context switching between threads is a lot simpler, due to the type of info you have to save/resume.

**Multithreaded server architecture**

Client -> Server -> Thread

Client makes request.  
Server creates new thread to service request  
Server listens for additional client requests.

A **program counter** points to where next line of code is. So multi-threads I like having multiple program counters running.

Each thread has their own set of registers.

**Benefits**

Responsiveness: you can continue doing X, while your email is busy sending. While a process is blocked, you can do other things in the system.

Resource Sharing: threads share resources of the process – easier than shared memory or message passing

Economy: Cheaper than process creation, context switching cheaper

Scalability: Process can make use of multiprocessor architecture.

**Concurrency vs Parallelism**

Concurrent execution: Single core, appears to be doing multiple things at a time, but is doing lightweight context switches between threads.

Parallelism execution: Multi-Core, actually simultaneously runs multiple threads.

Types of parallelism  
-**Data parallelism:** Break up the data among multiple threads/cores doing the same operation, then add everything back together. Different data, same operation. **-Task parallelism:** Distributing threads across cores, each thread performing unique operations. Different tasks that need to happen in parallel. Eg Word waiting for text input, and a task doing predictive text.

**Hardware threads:** Logical Processors, a “multhreading” that the CPU can use. Duplicates set of registers (more info in CHP1)

**#pragma omp parallel** – makes a piece of c++ code multithreaded