Operating Systems

Threads
Based on Ch. 4 of
OS Concepts by SGG

Is this about sewing

- Thread is a basic unit of computation
  - A process contains one or more threads
  - The all share their code, data, resources (open files, sockets, pipes, signals(?), etc)
  - They have separate thread ID, program counter, registers, stack.
- Many modern applications are multi threaded
- Multicore designs make multithreading attractive
- The model for modern GPUs is SIMT (Single Instruction Multiple Threads)

Kinds of Threads

- User level threads
  - Require no intervention by the OS
  - Could (in theory or in a dream) be written by any user; Very tricky to avoid blocking, starvation, etc
  - Low overhead (aka LWP: LightWeight Processes)
- Kernel level threads
  - Are treated (almost) as processes
  - Kernel provides proper blocking, scheduling etc
  - Can take advantage of multicore architectures

Examples

- A sophisticated application with a complex GUI may be multithreaded with one thread attending the GUI and the other the background computation.
- A web server may assign every request to a different thread to take advantage of the multicore CPU and let some threads work while the others are blocked. Same for other kinds of servers

Benefits

- Cheaper than processes (sharing resources, faster context switching)
  - By mixing user and kernel threads the cost goes even lower
- Can attend multiple tasks at once; responsiveness
- Faster and less restrictive communication
- Can make use of multiple cores

Parallelism vs Concurrency

- Two process/threads run in parallel if they run on multilpe CPUs/cores
- Two process/threads run concurrently if they all make progress (run in parallel or share a CPU by alternating on it)

Models

- All user level threads
  - One process (kernel level thread) handles all the threads (aka many to one)
  - A blocking system call would block all threads
    - Real world implementations provide simple workarounds
  - Cannot make use of multicore architectures
  - Examples: Sun LWP, Green Threads
- All kernel level threads
  - One user level thread per kernel thread (aka one to one)
  - Great, but can be costly in resources (memory or time)
  - Example: Linux threads

Models

- Mixed
  - A group of kernel level threads share several user level threads
  - There should be more user level threads than kernel ones
  - A user level thread may block
    - Either because executed a blocking system call (the corresponding kernel may or may not block)
    - Or because had to wait for another thread (the corresponding kernel should not block)
  - Solaris implements such a model

Libraries

- There can be two types of thread libraries
  - User space libraries
  - Kernel space libraries
- Pthreads can be either
- Windows libraries are kernel space
- Java threads use the threads of the underlying OS

Pthreads

- Main thread library of Linux/Unix world
- It is a specification
  - All pthread implementations can run the same code
  - Each is implemented differently
- Global data is available to all
- Address space is shared
- Threads are created and then are given a function pointer and data to start.
- Have join and other synch mechanisms

Windows threads

- Similar to pthreads
- It is an implementation rather than specification
- Has same access to global data
- Minor differences like create and start a thread in one step
- Comparable sync mechanism

Implicit Threading

- Classical multi threaded code is hard to write and the thread overhead can negate the benefits of threading
- Today we have multicore processors (and GPUs) with more than a thousand cores (processing elements)

Thread Pools

- One approach is to maintain a pool of threads that stay idle until they are needed
- The number of threads needed is estimated empirically or intuitively
- Solves the problem of overhead

OpenMP

- Open Multiprocessing makes use of multithreading with the help of directives like
  - #pragma omp parallel
- These directives tell the compiler to attempt to parallelize

Threading Issues

- What about fork and exec
  - Does the new program replace/duplicate all the threads of the process pod or just the calling one
- Who handles the signals
  - Deliver it to the thread to which applies
  - Deliver to every thread
  - Deliver it to some threads