What is a thread?

- A “thread” is a single sequential flow of control within a process.
- A thread runs within a process, sharing code with other threads.
- Multiple threads within a process can run concurrently.
- Thread also called “lightweight process” (just the execution context)

Benefits

- Responsiveness, faster context switch
- Resource Sharing, economy
- Parallelism, utilization of multicore and multiprocessor architectures
- Fits client/server model well
- Scalability

Sharing State

- Recall that a child forked from a parent process is a replica of the parent.
  - Modifications to the state (i.e., variables) are not visible to either side after forking.
  - Exchange of information must pass through an OS-provided channel.
- Threads are based on a model where threads spawned from a parent can see:
  - State residing within the parent.
  - State residing within other threads.
  - All through standard variables, not OS-provided mechanisms.
- From a programmability perspective, this makes building concurrent programs easier.

Threads and parallel systems

- Threads fit naturally onto parallel computers.
  - Such as multicore.
- Allows a programmer to build one program to use all of the cores.
  - Instead of a flock of coordinated processes.
- The thread abstraction facilitates:
  - Shared address spaces.
  - Independent units of execution that can be scheduled onto the parallel processing elements (cores).
**Multicore Programming**

- Multicore systems putting pressure on programmers, challenges include
  - Dividing activities
  - Balance
  - Data splitting
  - Data dependency
  - Testing and debugging

**Concurrent Execution on a Single-core System**

Given task T1, T2, T3, T4 capable of running concurrently.

On single core, must run one task after the other.

**Parallel Execution on a Multicore System**

- Core 1: T1, T3, T1, T3, T1 ...
- Core 2: T2, T4, T2, T4, T2 ...

**Multithreaded Server Architecture**

This can be done with parent and child processes, but less efficient, easier to share resources with threads.

**Kernel Threads**

- Supported by the kernel
- Examples
  - Windows XP/2000
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

**User Threads**

- Thread management done by user-level threads library. OS kernel does not manage threads (OS manages processes)
- User threads cheap, efficient.
- Creating threads, switching among threads, synchronizing threads done through procedure calls.
- Three primary user-level thread libraries:
  - POSIX Pthreads
  - Win32 threads
  - Java threads
Kernel v. User Threads

- Kernel threads not well-suited to fine-grained concurrency, too much overhead as switching between kernel threads involves invoking the kernel
- User-level threads allow fast switching among threads
- User-level threads operations 100x faster than kernel threads.
- However, because user-level threads are invisible to the OS, poor performance can result
  - Scheduling a process with idle threads
  - Blocking a process whose thread has requested I/O even though it has other runnable threads

Multithreading Models

- Many-to-One
- One-to-One
- Many-to-Many

Many-to-One

- Many user-level threads mapped to single kernel thread
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads

One-to-One

- Each user-level thread maps to kernel thread
- Examples
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and later

Many-to-One Model

One-to-one Model
Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9
- Windows NT/2000 with the ThreadFiber package

Two-level Model

- Similar to M:M, except that it allows a user thread to be bound to kernel thread
- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier

Thread Libraries

- Thread library provides programmer with API for creating and managing threads
  - create, fork, start, stop, block, yield, exit
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Pthreads

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)
Java Threads

- Java threads are managed by the JVM
- Typically implemented using the threads model provided by underlying OS
- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface

Operating System Examples

- Windows XP Threads
- Linux Thread

Windows XP Threads

- Implements the one-to-one mapping, kernel-level
- Each thread contains
  - A thread id
  - Register set
  - Separate user and kernel stacks
  - Private data storage area
- The register set, stacks, and private storage area are known as the context of the threads
- The primary data structures of a thread include:
  - ETHREAD (executive thread block)
  - KTHREAD (kernel thread block)
  - TEB (thread environment block)

Linux Threads

- Linux refers to them as tasks rather than threads
- Thread creation is done through clone() system call
- clone() allows a child task to share the address space of the parent task (process)