Chapter 4: Threads

- Overview
- Multicore Programming
- Multithreading Models
- Operating System Examples
Objectives

- To introduce the notion of a thread—a fundamental unit of CPU utilization that forms the basis of multithreaded computer systems
- To cover operating system support for threads in Windows and Linux

Motivation

- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded
Multithreaded Server Architecture

- (1) request
- (2) create new thread to service the request
- (3) resume listening for additional client requests

Benefits

- **Responsiveness** – may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing** – threads share resources of process, easier than shared memory or message passing
- **Economy** – cheaper than process creation, thread switching lower overhead than context switching
- **Scalability** – process can take advantage of multiprocessor architectures
Multicore Programming

- Multicore or multiprocessor systems putting pressure on programmers, challenges include:
  - Dividing activities
  - Balance
  - Data splitting
  - Data dependency
  - Testing and debugging
- Parallelism implies a system can perform more than one task simultaneously
- Concurrency supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

Multicore Programming (Cont.)

- Types of parallelism
  - Data parallelism – distributes subsets of the same data across multiple cores, same operation on each
  - Task parallelism – distributing threads across cores, each thread performing unique operation
- As # of threads grows, so does architectural support for threading
  - CPUs have cores as well as hardware threads
  - Consider Oracle SPARC T4 with 8 cores, and 8 hardware threads per core
Concurrent execution on a single-core system:

<table>
<thead>
<tr>
<th>Time</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T1</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Parallelism on a multi-core system:

- Core 1:
  | Time | T1 | T3 | T1 | T3 | T1 | ... |
  |      |    |    |    |    |    |     |

- Core 2:
  | Time | T2 | T4 | T2 | T4 | T2 | ... |
  |      |    |    |    |    |    |     |

Single and Multithreaded Processes

- Single-threaded process:
  - Code
  - Data
  - Files
  - Registers
  - Stack

- Multithreaded process:
  - Code
  - Data
  - Files
  - Registers
  - Registers
  - Registers
  - Stack
  - Stack
  - Stack

Thread
User Threads and Kernel Threads

- **User threads** - management done by user-level threads library
- Three primary thread libraries:
  - POSIX Pthreads
  - Windows threads
  - Java threads
- **Kernel threads** - Supported by the Kernel
- Examples – virtually all general purpose operating systems, including:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Multithreading Models

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

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on multicore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads

One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux
  - Solaris 9 and later
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 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
Operating System Examples

- Windows Threads
- Linux Threads

Windows Threads

- Windows implements the Windows API – primary API for Win 98, Win NT, Win 2000, Win XP, and Win 7
- Implements the one-to-one mapping, kernel-level
- Each thread contains
  - A thread id
  - Register set representing state of processor
  - Separate user and kernel stacks for when thread runs in user mode or kernel mode
  - Private data storage area used by run-time libraries and dynamic link libraries (DLLs)
- The register set, stacks, and private storage area are known as the context of the thread
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)
  - Flags control behavior

<table>
<thead>
<tr>
<th>flag</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLONE_FS</td>
<td>File-system information is shared.</td>
</tr>
<tr>
<td>CLONE_VM</td>
<td>The same memory space is shared.</td>
</tr>
<tr>
<td>CLONE_SIGCHLD</td>
<td>Signal handlers are shared.</td>
</tr>
<tr>
<td>CLONE_FILES</td>
<td>The set of open files is shared.</td>
</tr>
</tbody>
</table>

- `struct task_struct` points to process data structures (shared or unique)

End of Chapter 4