Chapter 4: Threads

- Overview
- Multithreading Models
- Thread Libraries
- Threading Issues
- Operating System Examples
- Windows XP Threads
- Linux Threads
Objectives

- To introduce the notion of a thread — a fundamental unit of CPU utilization that forms the basis of multithreaded computer systems
- To discuss the APIs for the Pthreads, Win32, and Java thread libraries
- To examine issues related to multithreaded programming
Single and Multithreaded Processes

Single-threaded process vs. multithreaded process.
Benefits

- Responsiveness
- Resource Sharing
- Economy
- Scalability
Multicore Programming

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

1. Request from client to server
2. Server creates new thread to service the request
3. Thread resumes listening for additional client requests
Concurrent Execution on a Single-core System

single core

| \( T_1 \) | \( T_2 \) | \( T_3 \) | \( T_4 \) | \( T_1 \) | \( T_2 \) | \( T_3 \) | \( T_4 \) | \( T_1 \) | ... |

\( T \) = Task

\( T \) = Task

Time line
Parallel Execution on a Multicore System
User Threads

- Thread management done by user-level threads library

- Three primary thread libraries:
  - POSIX Pthreads
  - Win32 threads
  - Java threads
Kernel Threads

- Supported by the Kernel

- Examples
  - Windows XP/2000
  - 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
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads
Many-to-One Model
One-to-One

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

user thread

kernel thread
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
Many-to-Many Model
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
Two-level Model
Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads
- 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
Threading Issues

- Semantics of `fork()` and `exec()` system calls
- Thread cancellation of target thread
  - Asynchronous or deferred
- Signal handling
- Thread pools
- Thread-specific data
- Scheduler activations
Semantics of fork() and exec()

- Does `fork()` duplicate only the calling thread or all threads?
Thread Cancellation

- Terminating a thread before it has finished
- Two general approaches:
  - Asynchronous cancellation terminates the target thread immediately
  - Deferred cancellation allows the target thread to periodically check if it should be cancelled
Signal Handling

- Signals are used in UNIX systems to notify a process that a particular event has occurred.
- A signal handler is used to process signals:
  1. Signal is generated by a particular event.
  2. Signal is delivered to a process.
  3. Signal is handled.
- Options:
  - Deliver the signal to the thread to which the signal applies.
  - Deliver the signal to every thread in the process.
  - Deliver the signal to certain threads in the process.
  - Assign a specific thread to receive all signals for the process.
Thread Pools

- Create a number of threads in a pool where they await work
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application(s) to be bound to the size of the pool
Thread Specific Data

- Allows each thread to have its own copy of data
- Useful when you do not have control over the thread creation process (i.e., when using a thread pool)
Scheduler Activations

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application.
- Scheduler activations provide upcalls - a communication mechanism from the kernel to the thread library.
- This communication allows an application to maintain the correct number kernel threads.
Operating System Examples

- Windows XP Threads
- Linux Thread
Windows XP Threads

ETHREAD

- thread start address
- pointer to parent process

KTHREAD

- scheduling and synchronization information
- kernel stack

TEB

- thread identifier
- user stack
- thread-local storage

kernel space | user space
# Linux Threads

<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_SIGHAND</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>
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 \textit{tasks} rather than \textit{threads}

- Thread creation is done through \texttt{clone()} system call

- \texttt{clone()} allows a child task to share the address space of the parent task (process)
End of Chapter 4