Chapter 4:  Threads
Chapter 4: Threads

- Overview
- Multithreading Models
- Thread Libraries
- Threading Issues
- Operating System Examples
- Linux Threads
Single and Multithreaded Processes

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

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

Thread

**References**
- Operating System Concepts
- Silberschatz, Galvin and Gagne
Multithread Programming

- Benefits
  - Responsiveness
  - Resource Sharing
  - Economy
  - Scalability

- Challenges include
  - Dividing activities
  - Balance
  - Data splitting
  - Data dependency
  - Testing and debugging
Multithreaded Server Architecture

1. The client sends a request to the server.

2. The server creates a new thread to service the request.

3. The server then resumes listening for additional client requests.
Concurrent Execution on a Single-core System

<table>
<thead>
<tr>
<th>Single core</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>
</table>

 tijd
Parallel Execution on a Multicore System

core 1

\[
\begin{array}{cccccc}
T_1 & T_3 & T_1 & T_3 & T_1 & \ldots \\
\end{array}
\]

core 2

\[
\begin{array}{cccccc}
T_2 & T_4 & T_2 & T_4 & T_2 & \ldots \\
\end{array}
\]

time
User Threads

- Thread management done by user-level threads library

- Three primary thread libraries:
  - POSIX Pthreads
  - Win32 threads
  - Java threads
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
  - 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
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 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
<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>
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)
Linux Thread

- Thread operations include thread creation, termination, synchronization (joins, blocking), scheduling, data management and process interaction.
- A thread does not maintain a list of created threads, nor does it know the thread that created it.
- All threads within a process share the same address space. They share:
  - Process instructions, Most data
  - open files (descriptors)
  - signals and signal handlers
  - current working directory, User and group id
- Each thread has a unique:
  - Thread ID, set of registers, stack pointer
  - stack for local variables, return addresses
  - signal mask,priority, Return value: errno
End of Chapter 4