Chapter 4: Threads

- Overview
- Multithreading Models
- Threading Issues
- Pthreads
- Windows XP Threads
- Linux Threads
- Java Threads

Single and Multithreaded Processes

- Benefits
  - Responsiveness
  - Resource Sharing
  - Economy
  - Utilization of MP Architectures

User and Kernel Threads

- User threads - Thread management done by user-level threads library.
- Kernel threads - Threads directly supported by the kernel.

Kernel Threads

- Examples
  - Windows XP/2000
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X
**Multithreading Models**

Mapping user threads to kernel threads:

- 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-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

- User thread
- Kernel thread

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

Java Threads

- Java threads are managed by the JVM
- Java threads may be created by:
  - Implementing the Runnable interface

```java
public interface Runnable { 
  public abstract void run();
}
```

Java Threads - Example Program

```java
class RunnableImpl implements Runnable { 
  private String name;
  private RunnableRunnable runThread;
  public RunnableImpl() { 
    this.name = "test value";
    this.runThread = new RunnableRunnable();
  }
  public void run() { 
    System.out.println("name: " + name);
    runThread.run();
  }
  public static void main(String[] args) { 
    new RunnableImpl();
  }
}
```
Java Thread States

Java Threads - Producer-Consumer

public class Factory

[Code snippet]

Java Threads - Producer-Consumer

[Code snippet]

Threading Issues

- Semantics of fork() and exec() system calls
- Thread cancellation
- 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

Deferred cancellation in Java

```java
class InterruptibleThread implements Runnable {

    private boolean interrupted = false;

    public void run() {
        while (!interrupted) { // Do some work here
            ...
        }
    }
}
P// No need for new thread, so...
Thread thread = new InterruptibleThread();
thread.start();
```

Signal Handling

- Signals are used in UNIX systems to notify a process that a particular event has occurred
- Procedure related to a signal
  - Signal is generated by particular event
  - Signal is delivered to a process
  - Signal is handled via a signal handler
- 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

Java provides 3 thread pool architectures:

1. **Single thread executor** - pool of size 1.
   ```java
   static ExecutorService newSingleThreadExecutor()
   ```

2. **Fixed thread executor** - pool of fixed size.
   ```java
   static ExecutorService newFixedThreadPool(int threads)
   ```

3. **Cached thread pool** - pool of unbounded size.
   ```java
   static ExecutorService newCachedThreadPool()
   ```
Thread Pools

A task to be serviced in a thread pool

```java
public class Task implements Runnable {
    public void run() {
        System.out.println("A task is now running.");
    }
}
```

Thread Pools

Creating a thread pool in Java

```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class ThreadPool {
    public static void main(String[] args) {
        ExecutorService pool = Executors.newFixedThreadPool(5);
        // Create the thread pool
        pool.execute(new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("A task is running.");
            }
        }));
        // Run each task using a thread in the pool
        for (int i = 0; i < 10; i++) {
            new Thread(new Runnable() {
                @Override
                public void run() {
                    System.out.println("A task is running.");
                }
            }).start();
        }
        // Shut down the pool. This shuts down the pool only
        // when all threads have completed.
        pool.shutdown();
    }
}
```

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)

Thread-specific data in Java.

```java
class Service {
    private static ThreadPool servicePool = new ThreadPool();
    public static void transaction(int id) {
        try {
            throw new Error("cannot happen");
        } catch (Error e) {
            throw new IOException(e);
        }
        // Get the error code for this transaction
    }
    public static void service() {
        servicePool.shutdown();
    }
}
```

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

Pthreads

- 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)
Windows XP Threads

- Implements the one-to-one mapping
- 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

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)

<table>
<thead>
<tr>
<th>Tag</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_SIG</td>
<td>Signals handled by the thread are shared.</td>
</tr>
<tr>
<td>CLONE_FILES</td>
<td>The set of open files is shared.</td>
</tr>
</tbody>
</table>