Threads

- Definition
- Multithreading Models
- Threading Issues
- Examples:
  - Pthreads
  - Solaris 2 Threads
  - Windows 2000 Threads
  - Linux Threads
  - Java Threads

What is a thread?

- A "thread" is a single sequential flow of control within a program, an "execution context".
- It has a beginning, a sequence, an end and a single point of execution at any given time.
- A thread by itself though is not a program, it runs within a program, sharing code with other threads.
- Lightweight process (just the execution context)

Single and Multithreaded Processes

- Thread management done by user-level threads library
- Examples
  - POSIX Pthreads
  - Mach C-threads
  - Solaris threads

Benefits

- Responsiveness, faster context switch
- Resource Sharing, economy
- Parallelism, Utilization of multiprocessor architectures
- Fits client/server model well

User Threads

- Supported by the OS Kernel
- Examples
  - Windows 95/98/NT/2000
  - Solaris
  - Tru64 UNIX
  - BeOS
  - Linux
Multithreading Models

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

Many-to-One

- Many user-level threads mapped to a single kernel thread.
- Used on systems that do not support kernel threads.

One-to-One

- Each user-level thread maps to a kernel thread.
- Examples:
  - Windows 95/98/NT/2000
  - OS/2

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 2
- Windows NT/2000 with the ThreadFiber package
Many-to-Many Model

Thread Issues
- Semantics of fork() and exec() system calls.
- Thread cancellation.
- Signal handling
- Thread pools
- Thread specific data

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 2 Threads

Solaris Process

Windows 2000 Threads
- Implements the one-to-one mapping.
- Each thread contains
  - a thread id
  - register set
  - separate user and kernel stacks
  - private data storage area
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).

Java Threads

- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface
- Java threads are managed by the JVM.

Concurrent Programming

Java Threads

- Java Tutorial
  - [http://java.sun.com/docs/books/tutorial/essential/threads/]
- Threads are the units of execution context in Java

Sequential Program

Concurrent Program

Java Thread States

- (a) Threads are always properly nested with co-begin statements
- (b) The fork/join mechanism is more general
  - In Java,
    - Fork by creating a Thread and executing start() on the thread
    - Join by executing join() on the thread

Fork/Join

- (a) Threads are always properly nested with co-begin statements
- (b) The fork/join mechanism is more general
  - In Java,
    - Fork by creating a Thread and executing start() on the thread
    - Join by executing join() on the thread
Creating threads

- There are two ways of creating threads in Java:
  - Extending the "Thread" class and over-riding its run method
  - Implementing the "Runnable" interface
- Since Java allows only single inheritance extension is possible only in cases where there is no super class.
- Implementing the interface requires the user to define methods declared in Runnable before using them.

An example using Runnable interface

```java
public class Clock extends Applet implements Runnable {
  private Thread clockThread = null;

  public void start() {
    if (clockThread == null) {
      clockThread = new Thread(this, "Clock");
      clockThread.start();
    }
  }

  public void run() {
    Thread myThread = Thread.currentThread();
    while (clockThread == myThread) {
      repaint();
      try {
        Thread.sleep(1000);
      } catch (InterruptedException e) {}
    }
  }

  // overrides Applet's stop method, not Thread's
  public void stop() {
    clockThread = null;
  }
}
```

Different stages in a Thread’s life

- Creating a thread:
  - clockThread = new Thread(this, "Clock");
- Starting a thread:
  - clockThread.start();
- Making a thread not runnable:
  - Putting a thread to sleep
  - Thread.sleep(1000);
  - Thread is waiting to be synchronized
  - yields();
  - Thread is blocked on I/O

Thread priority

- Since most computer configurations have a single CPU, threads actually run one at a time in such a way so as to provide an illusion of concurrency.
- Execution of multiple threads on a single CPU is called scheduling.
- The Java runtime supports a deterministic fixed priority scheduling algorithm.
- The algorithm schedules threads based on their priority relative to other runnable threads.

Different stages in a Thread’s life

- Stopping a thread:
  - A thread arranges for its own death by having a run method that terminates naturally.
  ```java
  public void run() {
    while (condition) { ...
  }
  ```
  ```java
  public void stop() {
    clockThread = null;
  }
  ```
- isAlive method:
  - The API for the Thread class includes a method called isAlive that returns TRUE when the thread has been started and not stopped. It returns FALSE when the thread is either a new thread or a dead one.
  ```java
  public boolean isAlive() {
  ```

Thread priority (contd…)

- When a Java thread is created it inherits its priority from the thread that created it unless modified using the setPriority method
- Thread priorities are integers ranging from MIN_PRIORITY to MAX_PRIORITY
- A chosen thread will run until
  - A higher priority thread becomes runnable (preemptive scheduling)
  - It yields or its run method exits
  - On systems that support time slicing, its time allotment has expired
Synchronizing threads

- In many interesting situations where separate concurrently running threads share data they must consider the state and activities of other threads i.e. must be synchronized (producer-consumer threads)
- The `wait` method relinquishes the lock held by a thread until the `notifyAll` method is invoked by a running thread which wakes up all waiting threads.

Grouping of threads

- Thread groups provide a mechanism for collecting multiple threads into a single object and manipulating them all at once.
- Every Java thread is a permanent member of a thread group.
- By default, a new thread is placed in the same group as the thread that created it.
- A thread can be made part of a non-default thread group only at creation time by using one of the three constructors in the `Thread` class.
- To find a thread's group we can use:
  ```java
  thread = myThread.getThreadGroup();
  ```