Multithreaded Processes

Dickinson College
Computer Science 354
Spring 2006

slides courtesy of Professor Grant Braught

Road Map

- Past:
  - What an OS is, why we have them, what they do.
  - Base hardware and support for operating systems
  - Process Management
  - Process Scheduling
- Present:
  - Multi-Threading
- Future:
  - Thread Synchronization
  - Memory management
  - Storage management
  - Protection and Security

Multithreading

- Some terminology
- Multithreading from the user perspective
  - What are threads and multithreading?
  - Threads in Java - pt 1.
  - Threads in Java - pt 2.
  - Examples and benefits of multithreading
- Multithreading from the system perspective

Terminology

- Some terminology that is useful in discussing the way processes and threads are executed:
  - Sequential Execution
  - Concurrent Execution
  - Parallel Execution

Sequential Execution

- **Sequential Execution**: execution events (A CPU operation or an I/O operation) occur one at a time and one after the other.

Concurrent Execution

- **Concurrent Execution**: when two or more execution events either actually or apparently occur simultaneously.

Parallel Execution

- **Parallel Execution**: a subset of concurrent execution in which the execution actually does happen simultaneously.
  - Examples:
    - A Disk I/O and a CPU operation
    - Several CPU operations on a Multiprocessor system

Multithreading from the User's Perspective

Thread

- A *thread* is a point of execution within a process.
- So a *multi-threaded process* has multiple points of concurrent execution within the process.
  - Think: Timesharing or multiprocessing among multiple execution points within a single program.
    - A traditional processes can be thought of as being a process with a single thread.

A First Java Thread Example

```
public class FirstThread {
   public static void main(String[] args) {
      System.out.println("Main thread running...");
      Thread t1 = new myThread(1);
      t1.start();
      System.out.println("Main thread finished...");
   }
}
```

```
class myThread extends Thread {
   private int id;
   public myThread(int id) {
      this.id = id;
   }
   public void run() {
      System.out.println("Thread " + id + "
   }
}
```

Threads Execute Concurrently

```
public class ConcurrentThreads {
   public static void main(String[] args) {
      System.out.println("Main starting...");
      Thread t1 = new myThread2(1);
      Thread t2 = new myThread2(2);
      t1.start();
      t2.start();
      for (int i=0; i<1000; i++) {
         System.out.println("Main running");
      }
      System.out.println("Main ending...");}
   }
}
```

```
class myThread2 extends Thread {
   private int id;
   public myThread2(int id) {
      this.id = id;
   }
   public void run() {
      for (int i=0; i<1000; i++) {
         System.out.println("Thread " + id
                            " running...");
      }
   }
```

Yield and Join

```
public class YieldJoin {
   public static void main(String[] args) {
      System.out.println("Main starting...");
      Thread t1 = new myThread3(1);
      Thread t2 = new myThread3(2);
      t1.start();
      t2.start();
      System.out.println("Main waiting...");
      try {
         t1.join();
         t2.join();
      }
      catch (InterruptedException e) {}
      System.out.println("Main ending...");
   }
}
```

```
class myThread3 extends Thread {
   private int id;
   public myThread3(int id) {
      this.id = id;
   }
   public void run() {
      for (int i=0; i<100; i++) {
         System.out.println("Thread " + id
                            " running...");
         Thread.yield();
      }
   }
}
```

Threads vs. Processes

- Threads executing within the same process share most of their address space.
  - ✓All threads in a process share the same:
    - ❖Code segment
    - ❖Data segment
    - ❖Heap
  - ✓However, each thread must have its own:
    - ❖Program counter
    - ❖Register values
    - ❖Stack segment

Threads Share Data and Heap

```
public class SharedAddressSpace {
  public static void main(String[] args) {
    int[] vals = {-1, -1, -1};
    Thread t0 = new myThread4(0,vals);
    Thread t1 = new myThread4(1,vals);
    Thread t2 = new myThread4(2,vals);
    t0.start(); t1.start(); t2.start();
    try {
      t0.join();  t1.join();  t2.join();
    }
    catch (InterruptedException e) {}
    System.out.println("vals[0] = " + vals[0]);
    System.out.println("vals[1] = " + vals[1]);
    System.out.println("vals[2] = " + vals[2]);
  }
}
```

```
class myThread4 extends Thread {
  private int id;
  private int[] array;
  public myThread4(int id, int[] array) {
    this.id = id;
    this.array = array;
  }
  public void run() {
    array[id] = id;
  }
}
```

Examples of Multi-Threading

- Many common processes make extensive use of multi-threading:
  - ✓Web browser
  - ✓Animation threads
  - ✓User interaction threads
  - ✓Database server
  - ✓Web server

Multithreading Benefits

- Anything that can be done with a multithreaded program can also be done:
  - ✓With a single threaded program
  - ✓With cooperating processes and IPC

Multithreading Benefits

- Compared to a single threaded version of the same program a multithreaded version may exhibit
  - ✓*better responsiveness*
  - ✓*improved performance*.

Multithreading Benefits

- Compared to an implementation using cooperating processes a multithreaded implementation will be:
  - ✓*more economical* in terms of system resource usage
  - ✓*more efficient* in terms of execution speed
    - ❖Creation
    - ❖Context Switching
    - ❖Communication

Multithreading vs. Cooperating Processes

- So when is it preferable to use multithreading and when is it preferable to use cooperating processes?

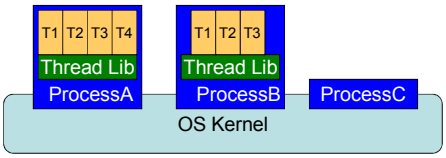
Multithreading from the OS Perspective

Thread Implementations

- Multithreading can be implemented at two distinct levels:
  - ✓ User Level Threads
  - ✓ Kernel Level Threads

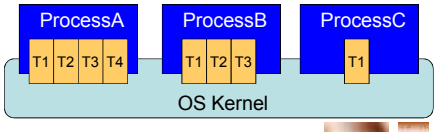
User Level Threads

- User level threads within a process are managed by a software library.
  - ✓ The OS unaware of user level threads.
  - ✓ Threads within a process:
    - ❖ Are non-preemptive
    - ❖ Suffer from one-block-all-block



Kernel Level Threads

- Kernel level threads are managed by the operating system.
  - ✓ The OS uses Thread Control Blocks (TCBs) to manipulate and schedule threads.
  - ✓ Threads within a process:
    - ❖ May be preemptive
    - ❖ Can be spread across multiple processors



Random OS Quote

- "The human mind ordinarily operates at only ten percent of its capacity — the rest is overhead for the operating system."

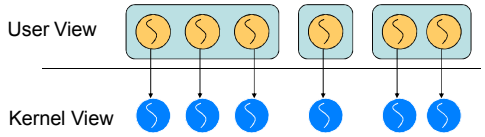
  Nicholas Ambrose

Threading Models

- The relationship between threads in a user program and threads in the kernel follow one of four models:
  - One-to-One
  - Many-to-One
  - Many-to-Many
  - 2-level model

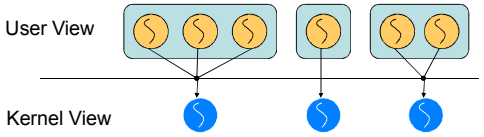
One-to-One

- In a one-to-one model every thread created by a user program maps to its own kernel level thread.
  - Pure kernel level threads.



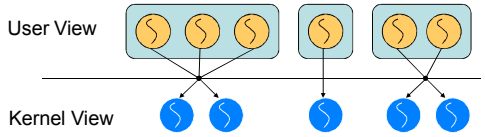
Many-to-One

- In a many-to-one model, multiple user level threads are mapped to a single kernel level thread.
  - Pure user level threads.



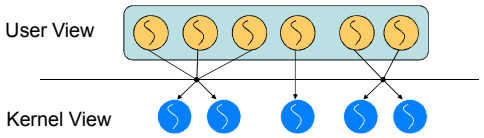
Many-to-Many

- In a many-to-many model, a thread library multiplexes a collection of user level threads onto an equal or smaller number of kernel threads.



2-Level Model

- The 2-Level Model combines a many-to-many model with the one-to-one model.
  - Multiple many to many mappings can exist within a process.
    - Including the special case where one user thread can be bound to one kernel thread.



Thread Implementations

- Each OS and language has its own implementation of threads:
  - Win32
  - POSIX Threads (pThreads)
  - Green threads
  - GNU portable threads
  - Java