CS 580 Client-Server Programming
Spring Semester, 2009
Doc 6 Threads
11 Feb, 2009

Copyright ©, All rights reserved. 2009 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.
References

The Java Programming Language, 2nd Ed. Arnold & Gosling, Addison-Wesley, 1998


Java 1.5.0 on-line documentation http://java.sun.com/j2se/1.5.0/docs/api/

Programming Ruby, 2'ed Thomas, Chapter 11 Threads and Processes, Thread class documentation (pp 633-639 or http://www.rubycentral.com/ref/ref_c_thread.html)

Reading

Java Network Programming, 3nd Ed., Harold, Chapter 5. (Java)

Programming Ruby, 2'ed Thomas, Chapter 11 Threads and Processes
Concurrent Programming

Safety

Liveness

Nondeterminism

Communication
Processes verses Threads

Processes (Heavy Weight)
Child process gets a copy of parent’s variables
Relatively expensive to start
No concurrent access to variables

Thread (Light Weight Process)
Child process shares parents variables
Relatively cheap to start
Concurrent access to variables is an issue
Creating Threads by Inheritance

class ExtendingThreadExample extends Thread {
    public void run() {
        for ( int count = 0; count < 4; count++)
            System.out.println( "Message " + count + " From: Mom" );
    }

    public static void main( String[] args ) {
        ExtendingThreadExample parallel =
            new ExtendingThreadExample();
        System.out.println( "Create the thread" );
        parallel.start();
        System.out.println( "Started the thread " + parallel.getId() );
        System.out.println( "End" );
    }
}
class SecondMethod implements Runnable {
    public void run() {
        for (int count = 0; count < 4; count++)
            System.out.println("Message "+count +" From: Dad");
    }

    public static void main(String[] args) {
        SecondMethod notAThread = new SecondMethod();
        Thread parallel = new Thread(notAThread);
        System.out.println("Create the thread");
        parallel.start();
        System.out.println("Started the thread");
        System.out.println("End");
    }
}
public class WithNames implements Runnable {
    public void run() {
        for (int count = 0; count < 2; count++)
            System.out.println("Message "+count +
                                " From: " + Thread.currentThread().getName());
    }

    public static void main( String[] args ) {
        Thread a = new Thread(new WithNames(),
                                "Mom ");
        Thread b = new Thread(new WithNames(),
                                "Dad ");

        System.out.println("Create the thread");
        a.start();
        b.start();
        System.out.println("End ");
    }
}
Ruby Threads

```ruby
a = Thread.new { 4.times { |k| puts k } }
a.join
```

Output

```
0
1
2
3
```

```ruby
x = 5
a = Thread.new(x) do |size|
  size.times { |k| puts k }
end
a.join
```

Output

```
0
1
2
3
5
```
public class SimpleThread extends Thread {
    private int maxCount = 32;

    public SimpleThread( String name ) {
        super( name );
    }

    public SimpleThread( String name, int repetitions ) {
        super( name );
        maxCount = repetitions;
    }

    public SimpleThread( int repetitions ) {
        maxCount = repetitions;
    }

    public void run() {
        for ( int count = 0; count < maxCount; count++ ) {
            System.out.println( count + " From: " + getName() );
        }
    }
}

For Future Examples
public class RunSimpleThread {
    public static void main( String[] args ) {
        SimpleThread first = new SimpleThread( 5 );
        SimpleThread second = new SimpleThread( 5 );
        first.start();
        second.start();
        System.out.println( "End" );
    }
}

Output On Rohan

End
0 From: Thread-0
1 From: Thread-0
2 From: Thread-0
0 From: Thread-1
1 From: Thread-1
2 From: Thread-1
3 From: Thread-0
3 From: Thread-1
4 From: Thread-0
4 From: Thread-1

Java on a Solaris machine with multiple processors can run threads on different processors
Ruby

```ruby
a = Thread.new do
  5.times { |k| puts "a #{k}" }
end

b = Thread.new do
  5.times { |k| puts "b #{k}" }
end

a.join
b.join
```

Output

```
a 0
b 0
a 1
b 1
a 2
b 2
a 3
b 3
a 4
```
Thread Scheduling

Priorities

Time-slicing
Priorities

Each thread has a priority

If there are two or more active threads
   If one has higher priority than others
   The higher priority thread is run until it is done or not active

### Java Thread Priorities

<table>
<thead>
<tr>
<th>java.lang.Thread field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread.MAX_PRIORITY</td>
<td>10</td>
</tr>
<tr>
<td>Thread.NORM_PRIORITY</td>
<td>5</td>
</tr>
<tr>
<td>Thread.MIN_PRIORITY</td>
<td>0</td>
</tr>
</tbody>
</table>

### Ruby Thread Priorities

Any float between
- -2147483649
  2147483648

May be machine dependent
Java Priority

public class PriorityExample {
    public static void main( String[] args ) {
        SimpleThread first = new SimpleThread( 5 );
        SimpleThread second = new SimpleThread( 5 );
        second.setPriority( 8 );
        first.start();
        second.start();
        System.out.println( "End" );
    }
}

On Single Processor
0 From: Thread-5
1 From: Thread-5
2 From: Thread-5
3 From: Thread-5
4 From: Thread-5
0 From: Thread-4
1 From: Thread-4
2 From: Thread-4
3 From: Thread-4
4 From: Thread-4
End

On Multiple Processor Rohan
End
Ruby Priority

a = Thread.new do
  sleep
  5.times {|k| puts "a #{k}"}
end

b = Thread.new do
  sleep
  5.times {|k| puts "b #{k}"}
end

b.priority=-1
a.priority=-2
a.run
sleep(0.003)
b.run

a.join
b.join

Output

a 0
b 0
b 1
b 2
b 3
b 4
a 1
a 2
a 3
a 4
Threads Run Once

Can't restart a thread

```java
public class RunOnceExample extends Thread {
    public void run() {
        System.out.println( "I ran" );
    }

    public static void main( String args[] ) throws Exception {
        RunOnceExample onceOnly = new RunOnceExample();
        onceOnly.setPriority( 6 );
        onceOnly.start();

        System.out.println( "Try restart" );
        onceOnly.start(); // Causes Exception

        System.out.println( "The End" );
    }
}
```
Time-Slicing

A thread is run for a short time slice and suspended,
It resumes only when it gets its next "turn"

 Threads of the same priority share turns

Non time-sliced threads run until:
  They end
  They are terminated
  They are interrupted
  Higher priority threads interrupts lower priority threads
  They go to sleep
  They block on some call
  Reading a socket
  Waiting for another thread

Java spec allows time-sliced or non-time-sliced threads

Ruby docs don't talk about this
Testing for Time-slicing

If time-sliced output will be mixed

```java
public class InfinityThread extends Thread {
    
    public void run() {
        while (true) {
            System.out.println("From: " + getName());
        }
    }

    public static void main(String[] args) {
        InfinityThread first = new InfinityThread();
        InfinityThread second = new InfinityThread();
        first.start();
        second.start();
    }
}
```

```ruby
a = Thread.new do
  10.times { |k| puts "a #{k}" }
end

b = Thread.new do
  10.times { |k| puts "b #{k}" }
end

a.join
b.join
```

If time-sliced output will be mixed
Java user & daemon Threads

Daemon thread
Expendable
When all user threads are done
  the program ends
  all daemon threads are stopped

User thread
Not expendable
Execute until
  Their run method ends or
  An exception propagates beyond the run method.
When a Java Program Ends

Runtime.exit(int) has been called and the security manager permits the exit operation to take place.

or

Only daemon threads are running
public class DaemonExample extends Thread {
    public static void main( String args[] ) {
        DaemonExample shortLived = new DaemonExample( );
        shortLived.setDaemon( true );
        shortLived.start();
        System.out.println( "Bye" );
    }

    public void run() {
        while (true) {
            System.out.println( "From: " + getName() );
            System.out.flush();
        }
    }
}

Output
From: Thread-0 (Repeated many times)
Bye
From: Thread-0 (Repeated some more, then the program ends)
Ruby Threads are daemon threads

Using Java terminology all Ruby threads are daemon threads
Thread States

Executing

Only one thread per processor can be running at a time

Runnable

A thread is ready to run but is not currently running

Not Runnable

A thread that is suspended or waiting for a resource
Yield

Allow another thread of the same priority to run
Thread is still runnable

public class YieldThread extends Thread {
    public void run() {
        for (int count = 0; count < 4; count++) {
            System.out.println(count + " From: " + getName());
            yield();
        }
    }
}

public static void main(String[] args) {
    YieldThread first = new YieldThread();
    YieldThread second = new YieldThread();
    first.setPriority(1);
    second.setPriority(1);
    first.start();
    second.start();
    System.out.println("End");
}

Output (Explain this)
0 From: Thread-0
0 From: Thread-1
1 From: Thread-0
1 From: Thread-1
2 From: Thread-0
2 From: Thread-1
3 From: Thread-0
3 From: Thread-1

Allow another thread of the same priority to run
Thread is still runnable
Ruby pass
Allow another thread of the same priority to run
Thread is still runnable

```ruby
a = Thread.new do
  10.times do |k|
    puts "a #{k}"
    Thread.pass
  end
end

b = Thread.new do
  10.times do |k|
    puts "b #{k}"
  end
end

a.join
b.join
```

Output
```
a 0b 0
b 1a 1
b 2a 2
b 3
a 3b 4
a 4b 5
b 6a 5
b 7a 6
b 8a 7
b 9
a 8
a 9
```
public class NiceThread extends Thread {
    public void run() {
        try {
            System.out.println( "Thread started" );
            sleep( 5 );
            System.out.println( "From: " + getName() );
            System.out.println( "Clean up operations" );
        }
        catch ( InterruptedException interrupted ) {
            System.out.println( "In catch" );
        }
    }
    
    public static void main( String args[] ) { 
        NiceThread missManners = new NiceThread();
        missManners.start();
        System.out.println( "Main after start" );
    }
}
Java sleep

Put **calling** thread in not-runnable state for specified milliseconds

```java
public class NiceThread extends Thread {
    public void run() {
        System.out.println( "Thread started" );
        System.out.println( "From: " + getName() );
        System.out.println( "Clean up operations" );
    }

    public static void main( String args[] ) throws InterruptedException {
        NiceThread missManners = new NiceThread();
        missManners.start();
        missManners.sleep(50);  //Who is sleeping
        System.out.println( "Main after start" );
    }
}
```

Output

Thread started
From: Thread-0
Clean up operations
Main after start
Ruby sleep

a = Thread.new do
  sleep
  5.times { |k| puts "a #{k}" }
end

b = Thread.new do
  sleep
  5.times { |k| puts "b #{k}" }
end

b.priority = -1
a.priority = -2
a.run
sleep(0.003)
b.run

a.join
b.join

Put calling thread in not-runnable state for specified seconds

Time can be a float

sleep(0) & sleep put thread to sleep indefinitely
Java deprecated Thread methods

The following Thread methods are not thread safe

suspend
resume
stop
destroy
Ruby exit & kill Class Methods

```
kill -Terminate given thread

count = 0
a = Thread.new { loop { count += 1}}
sleep(0.1)
Thread.kill(a)
puts count
puts a.alive?
```

Output

```
56946
false
```

```
exit -Terminate current thread

count = 0
a = Thread.new do
  loop do
    count += 1
    Thread.exit if count > 5000
  end
end
sleep(0.1)
puts count
puts a.alive?
```

Output

```
5000
false
```
Ruby exit, kill, terminate - Instance Methods

exit, kill, terminate -> same as Thread.kill

count = 0
a = Thread.new { loop { count += 1}}
sleep(0.1)
a.kill
puts count
puts a.alive?

count = 0
a = Thread.new { loop { count += 1}}
sleep(0.1)
a.exit
puts count
puts a.alive?

count = 0
a = Thread.new { loop { count += 1}}
sleep(0.1)
a.terminate
puts count
puts a.alive?