Threads in Java

Department of Computer Science
University of Maryland, College Park
Problem

• Multiple tasks for computer
  • Draw & display images on screen
  • Check keyboard & mouse input
  • Send & receive data on network
  • Read & write files to disk
  • Perform useful computation (editor, browser, game)

• How does computer do everything at once?
  • Multitasking
  • Multiprocessing
Multitasking (Time-Sharing)

• Approach
  • Computer does some work on a task
  • Computer then quickly switch to next task
  • Tasks managed by operating system (scheduler)
  • Computer *seems* to work on tasks concurrently
  • Can improve performance by reducing waiting
Multitasking Can Aid Performance

• Single task

- Total Execution Time = 7 seconds
- Busy

- 1 sec

- Total Time Executing Code: 4 seconds
- Total Time Waiting: 3 seconds
- Time Executing Code: 57%  Time Waiting: 43%

• Two tasks

- Total Time Executing Code: 8 seconds
- Total Time Waiting: 0 seconds
- Time Executing Code: 100%  Time Waiting: 0%
Multiprocessing

• Approach
  • Multiple processing units
  • **Computer works on several tasks in parallel**
  • Performance can be improved

Dual-core AMD Athlon X2

32 processor Pentium Xeon

4096 processor Cray X1

Beowulf computer cluster (Borg, 52-node cluster used by McGill University Image/Info from Wikipedia)
Perform Multiple Tasks Using Processes

- **Process**
  - Definition - executable program loaded in memory
  - Has own *address space*
  - Address space - Variables & data structures (in memory)
  - Each process may execute a different program
  - Communicate via operating system, files, network
  - A process may contain multiple threads
Perform Multiple Tasks Using Threads

- Thread (“lightweight process”)
  - Definition → sequentially executed stream of instructions
  - Has own execution context
    - Program counter, call stack (local variables)
  - Communicate via shared access to data
  - Also known as “lightweight process”
  - Let’s see how memory is organized for a threaded environment
  - Diagram
Motivation for Multithreading

- Captures logical structure of problem
  - May have concurrent interacting components
  - Can handle each component using separate thread
  - Simplifies programming for problem

- Example

Web Server uses threads to handle ... Multiple simultaneous web browser requests
Motivation for Multithreading

• Better utilization of hardware resources
  • When a thread is delayed, execute other threads
  • Given extra hardware, execute threads in parallel
  • Reduce overall execution time

• Example

Multiple simultaneous web browser requests... Handled faster by multiple web servers
Concurrent Programming

- Concurrent programming
  - Writing programs divided into independent tasks
  - Tasks may be executed in parallel on multiprocessors
Creating Threads in Java

- Two approaches to create threads
  - Extending Thread class (NOT RECOMMENDED)
  - Runnable interface approach (PREFERED)

- Approach 1: Extending Thread class
  - We override the Thread class run() method
  - The run() method defines the actual task the thread performs

- Example:
  ```java
  public class MyT extends Thread {
    public void run() {
      ...
      // Defines task for the thread
    }
  }

  MyT t = new MyT(); // Create thread
  t.start(); // Thread gets in line waiting to be executed
  ...
  ```

- Example: message, messageThreadExtends packages
Creating Threads in Java

• **Approach 2: Runnable Interface**
  
  • Define a class (worker) that implements the **Runnable** interface
  
  ```java
  public interface Runnable {
    public void run(); // work done by thread
  }
  ```
  
  • Create thread to execute the **run()** method
    
    • **Alternative 1**: Create thread object and pass worker object to Thread constructor
    
    • **Alternative 2**: Hand worker object to an executor
    
  • **Example**:
    
    ```java
    public class Worker implements Runnable {
      public void run() { // work for thread
      }
    }
    
    Thread t = new Thread(new Worker()); // Create thread
    t.start(); // Thread gets in line waiting to be executed
    ... 
    ```
  
  • **Example**: message, messageThreadRunnable packages
Why Extending Thread Not Recommended?

• Not a big problem for getting started
  • But a bad habit for industrial strength development
• Methods of worker and Thread class intermixed
• Hard to migrate to more efficient approaches
  • Thread Pools
Thread Class

public class Thread extends Object implements Runnable {
    public Thread();
    public Thread(String name);  // Thread name
    public Thread(Runnable R);
    public Thread(Runnable R, String name);

    public void run();  // work for thread
    public void start(); // thread gets in line so it eventually it can run
...
}

© Department of Computer Science UMD
More Thread Class Methods

```java
public class Thread extends Object {
    ...
    public static Thread currentThread()
    public String getName()
    public void interrupt()  // alternative to stop (deprecated)
    public boolean isAlive()
    public void join()
    public void setDaemon()
    public void setName()
    public void setPriority()
    public static void sleep()
    public static void yield()
}
```
Creating Threads in Java

• Note
  • Thread eventually starts executing only if start() is called
  • Calling start() does not mean the thread will start executing immediately

- Runnable is an interface
  • Therefore, it can be implemented by any class
  • A class can implement the interface, but not used for threading

• Do not call the run method directly
  • If using class instance as a thread
Threads – Thread States

- Java thread can be in one of these states
  - **New** → thread allocated & waiting for start()
  - **Runnable** → thread can begin execution
  - **Running** → thread currently executing
  - **Waiting/Blocked** → thread waiting for event (I/O, etc.)
  - **Terminated/Dead** → thread finished/exited

- Transitions between states caused by
  - Invoking methods in class Thread
    - `new()`, `start()`, `yield()`, `sleep()`, `wait()`, `notify()`…
  - Other (external) events
    - Scheduler, I/O, returning from `run()`…

- In Java, states are defined by `Thread.State`
Threads – Thread States

- State diagram

new → start → runnable

new

scheduler

runnable

yield, time slice

IO, sleep expired, join complete

running

terminate

IO, sleep, wait, join

blocked

dead

notify, notifyAll, IO complete

Running is a logical state → indicates runnable thread is actually running
Reference

- [https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html](https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html)