CMSC 132: OBJECT-ORIENTED PROGRAMMING II

Threads in Java

Department of Computer Science
University of Maryland, College Park

Problem

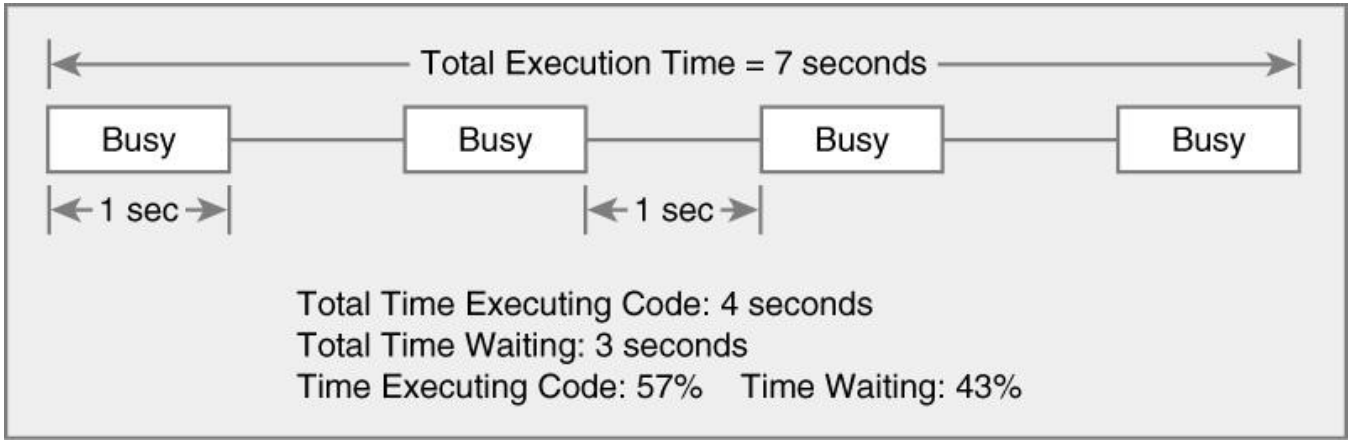
- Multiple tasks for computer
  - Draw & display images on screen
  - Check keyboard & mouse input
  - Send & receive data on network
  - Read & write files to disk
  - Perform useful computation (editor, browser, game)
- How does computer do everything at once?
  - Multitasking
  - Multiprocessing

Multitasking (Time-Sharing)

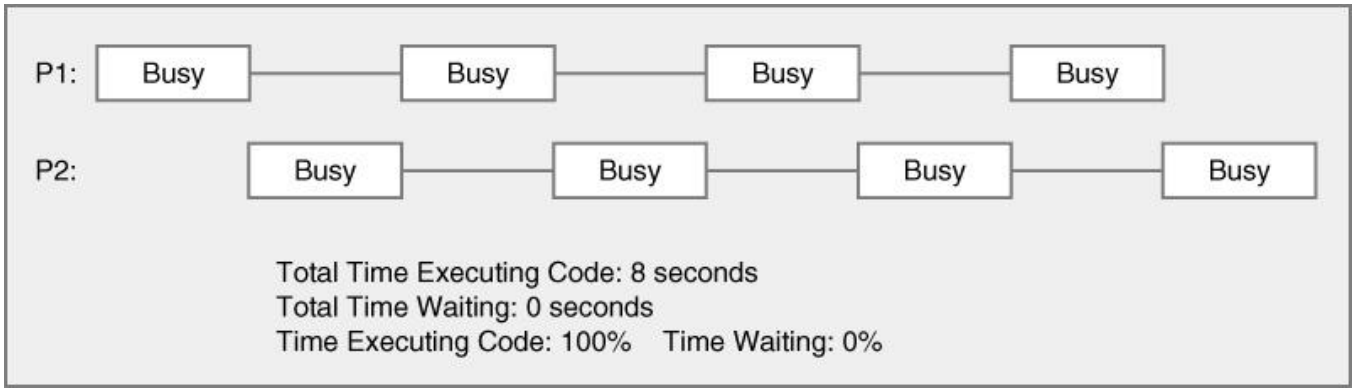
- Approach
  - Computer does some work on a task
  - Computer then quickly switch to next task
  - Tasks managed by operating system (scheduler)
- Computer seems to work on tasks concurrently
- Can improve performance by reducing waiting

Multitasking Can Aid Performance

- Single task



- Two tasks



Multiprocessing (Multithreading)

- Approach
  - Multiple processing units (multiprocessor)
  - 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

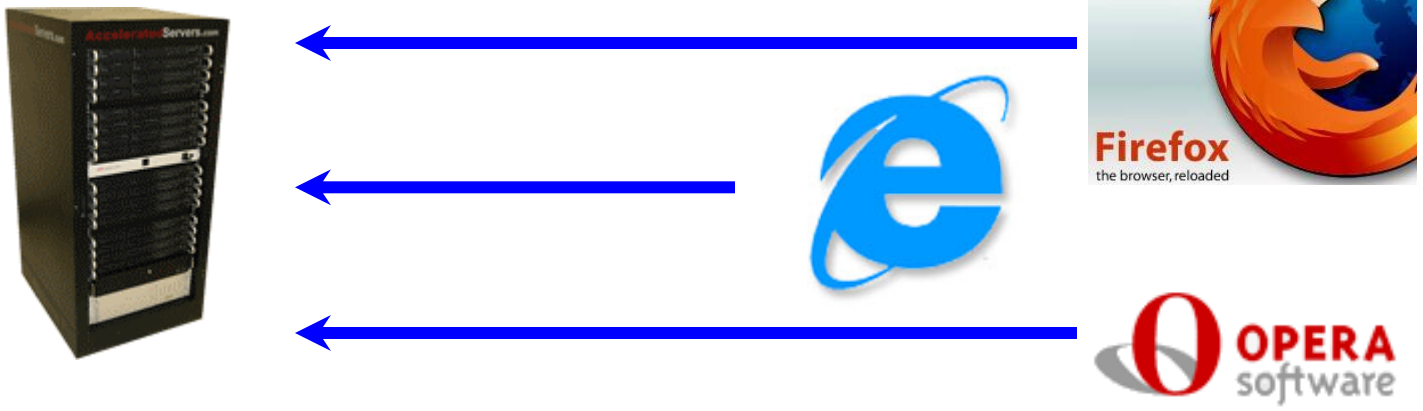
- Process
  - Definition → executable program loaded in memory
  - Has own address space
    - Variables & data structures (in memory)
  - Each process may execute a different program
  - Communicate via operating system, files, network
  - May contain multiple threads

Perform Multiple Tasks Using Threads

- Thread
  - 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
    - http://blog.codecentric.de/wp-content/uploads/2009/12/java-memory-architecture.jpg

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 utilize hardware resources
  - When a thread is delayed, compute other threads
  - Given extra hardware, compute threads in parallel
  - Reduce overall execution time
- Example

Multiple simultaneous web browser requests…

Handled faster by multiple web servers

Programming with Threads

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

- Multithreading
  - Executing program with multiple threads in parallel
  - Special form of multiprocessing

Creating Threads in Java

- Two approaches to create threads
  - Extending Thread class (**NOT RECOMMENDED**)
  - Runnable interface approach (**PREFERED**)
- Approach 1: Extending Thread class
  - We overload the Thread class run() method
  - The run() methods defines the actual task the thread performs
  - Example

```
public class MyT extends Thread {
    public void run( ) {
        …                       // work for thread
    }
}
MyT t = new MyT( ) ;    // create thread
t.start( );             // begin running thread
…                       // thread executing in parallel
```

- **Example:** message, messageThreadExtends packages

Creating Threads in Java

- Approach 2: Runnable Interface
  - Define a class (worker) that implements the Runnable interface

```
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

```
public class Worker implements Runnable {
    public void run( )  { // work for thread }
}
Thread t = new Thread(new Worker( ));  // create thread
t.start();                             // begin running thread
…                                      // thread executing in parallel
```

- **Example:** message, messageThreadRunnable packages

Why Extending Thread Approach 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

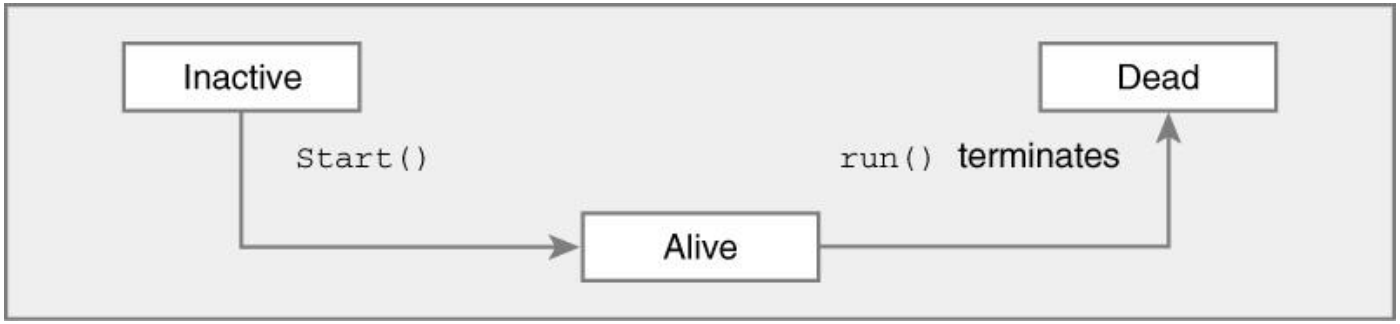
```
public class Thread extends Object implements Runnable {
    public Thread();
    public Thread(String name);   // Thread name
    public Thread(Runnable R);
    public Thread(Runnable R, String name);

    public void run();     // if no R, work for thread
    public void start();   // begin thread execution
    ...
}
```

More Thread Class Methods

```
public class Thread extends Object {
    …
    public static Thread currentThread()
    public String getName()
    public void interrupt()
    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



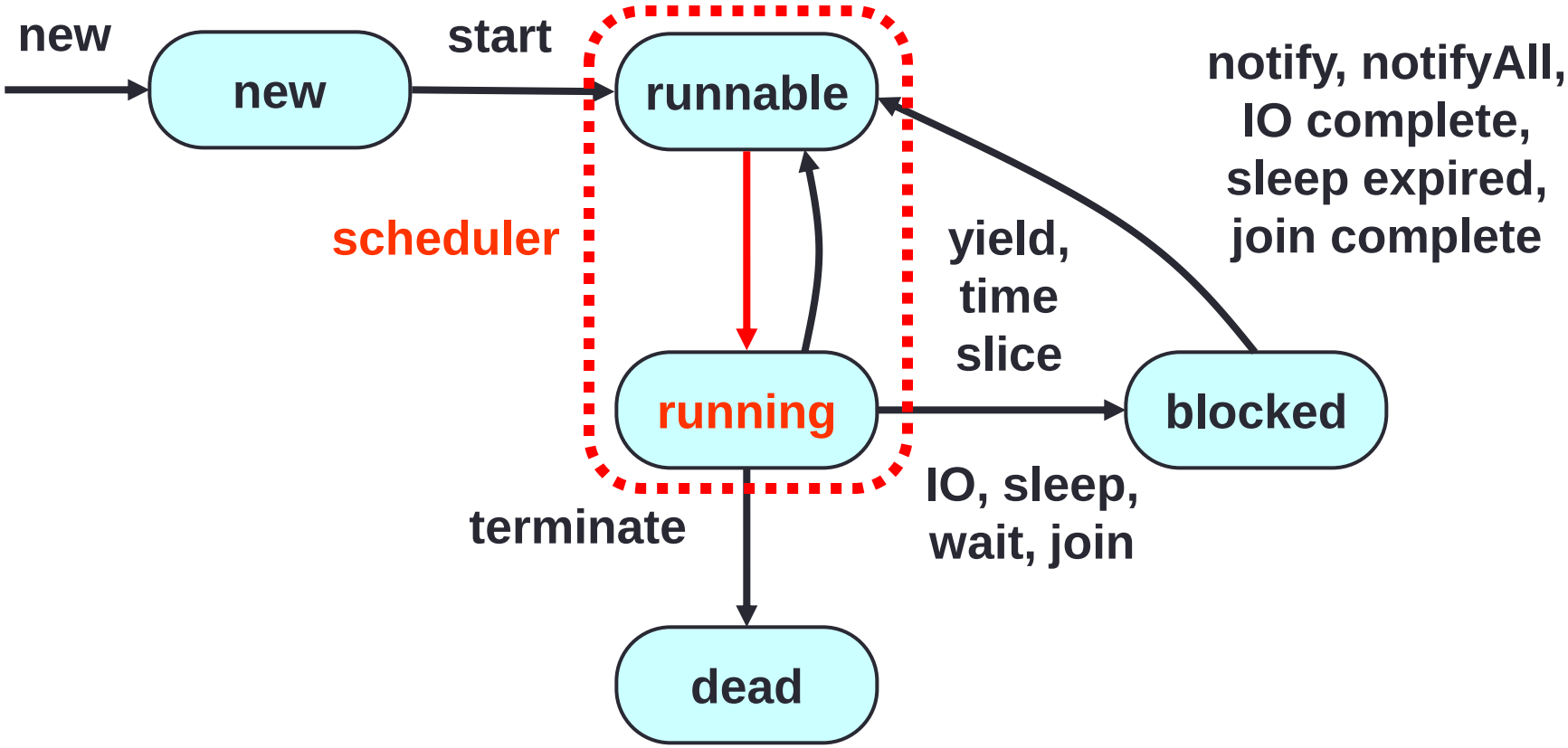
  - Runnable is interface
    - So it can be implemented by any class
    - Required for multithreading in applets
  - Do not call the run method directly

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
  - Blocked → thread waiting for event (I/O, etc.)
  - Dead → thread finished
- 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 defined by Thread.State

Threads – Thread States

- State diagram



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