SOFE 4790U DISTRIBUTED SYSTEMS

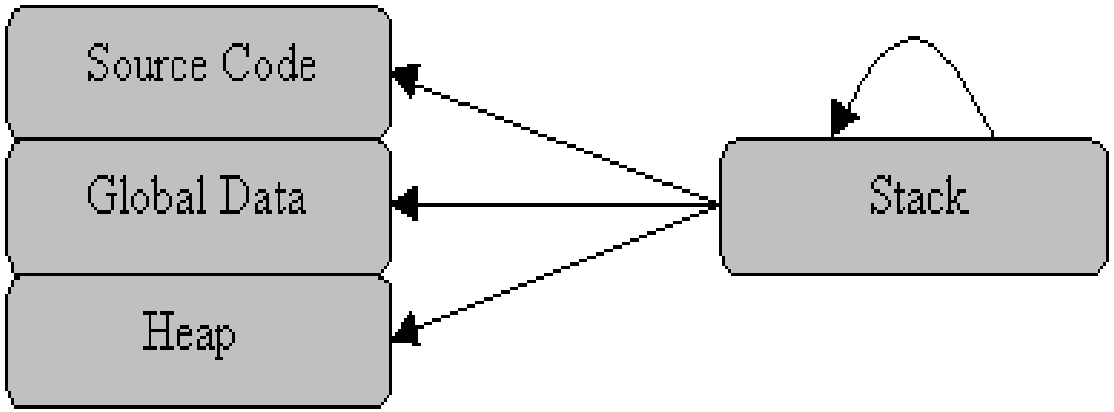
MULTITHREADING PROGRAMMING WITH JAVA

Fall 2020

Qusay H. Mahmoud, Ph.D.

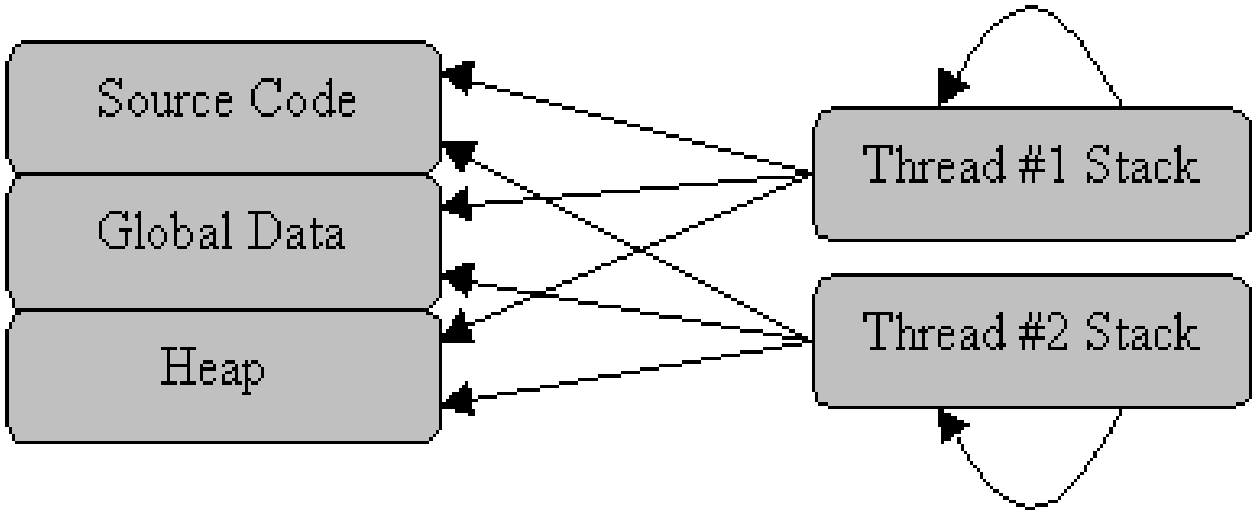
Sequential Programs

- A program with a single flow of control is called a **sequential** program
- A program has four parts: *source code*, *global data*, *heap*, and *stack*



Concurrent Programs

- A program with multiple points of execution is called a **concurrent** program



Tasks and Processes

- A task is the execution of a sequential program (or a sequential program within a concurrent program)
- A process is used in Operating Systems (OS) as a unit of resource allocation for CPU and memory
- A traditional OS process has a single thread of control (i.e. no internal concurrency)
- Modern OSs allow a process known as a heavyweight process (i.e. with multiple threads of control – concurrency within the process)

Heavyweight vs. Lightweight

- Each thread of control within a heavyweight process is known as a lightweight process
  - Because it shares the same memory
- Multiple threads of a heavyweight process can access shared data in the process's memory
- Access must be synchronized
- "heavy" and "light" refers to the context-switching overhead (CPU and memory allocation vs. CPU allocation)

What is a Thread?

- The term *thread* derives from the phrase ***thread of execution*** in operating systems
- It is a *lightweight process*
- Threads can create other threads and kill them
- Newly created threads will run in the same address space allowing them to share data
- They have been around for quite some time
- They are built-in into Java
- Java made the use of them easy and productive
  - If you haven't programmed with Threads, check the Java Tutorial

Benefits of Threads

- The ability to perform multiple tasks simultaneously
- Allow us to take advantage of computers with multiple CPUs
- Other benefits
  - Increase application throughput
  - Responsiveness
  - The ability to use system's resource efficiently
- Multi-threaded servers and clients

Programming with Threads (Java)

- Creating and Starting Threads
- Putting a Thread to Sleep
- Controlling Threads
- Thread Priorities
- Pitfalls of Threads
- Synchronization
- Producer/Consumer
- Scheduling

Creating and Starting Threads (1)

- There are two ways to create a thread in Java
  - Extending the **`Thread`** class

```
class MyThread extends Thread {
    ....
    public void run() {
      ....
    }
    public static void main(String argv[]) {
      MyThread t1 = new MyThread();
      t1.start();
    }
}
```

Creating and Starting Threads (2)

- The other way of creating a thread in Java is
  - By implementing the **`Runnable`** interface

```
class MyThread implements Runnable {
   public void run() {
      ....
   }
   public static void main(String argv[]) {
      MyThread s = new MyThread();
      Thread t1 = new Thread(s);
       t1.start();
   }
}
```

Creating and Starting Threads (3)

- Examples:
  - MyThread.java (extending Thread)
  - MyThread2.java (implementing Runnable)
  - Counter.java

Putting a Thread to Sleep

- You may pause a thread for a specific period of time by putting it to sleep using **`sleep()`**

```
try {
   Thread.sleep(4000); // 4 seconds
} catch (InterruptedException ie) {
   ie.printStackTrace();
}
```

- The argument to sleep specifies the number of milliseconds the thread will sleep for

Controlling Threads

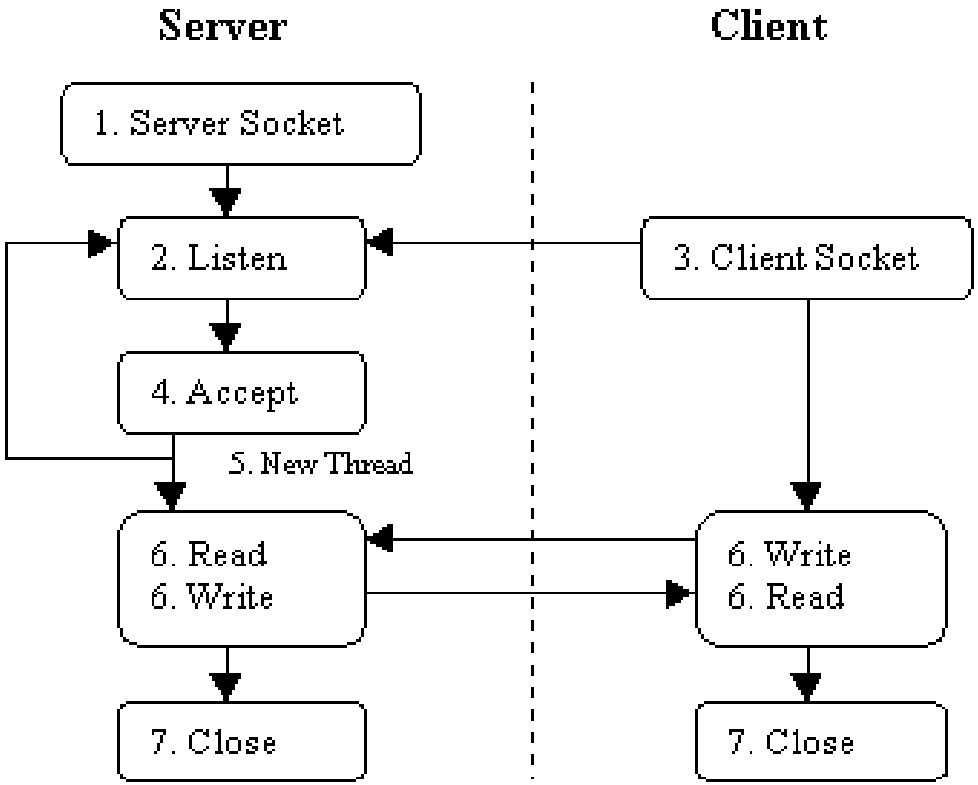
- Do not use: stop(), suspend(), resume()
- These methods have been deprecated in Java 2 and they should not be used
- Basically they are not thread-safe

Thread Priorities

- Threads will normally be competing for processor time
- Time-critical tasks with hard deadlines can be given a higher priority than less critical tasks
- The Thread class defines three constants:
  - MAX_PRIORITY (10)
  - MIN_PRIORITY (1)
  - NORM_PRIORITY (the default 5)
- Use getPriority() and setPriority()

Multi-threaded Servers

- A server should be able to serve multiple clients concurrently...



Threads Pitfalls

- One pitfall of threads is data sharing
  - Examples: Alice and Bob are sharing a checkbook

```
int balance;
boolean withdraw(int amount);
   if (balance - amount >= 0) {
     balance = balance - amount;
     return true;
   }
   return false;
}
```

Threads Pitfalls...

- If Alice and Bob are executing the code segment simultaneously, we get:

Alice	Bob	Balance
If (80 - 50 >= 0)		80
	If (80 - 70 >= 0)	80
Balance = Balance - 50		30
	Balance = Balance - 70	-40

Synchronization

- Mutual Exclusion (preventing simultaneous access to shared data) can be accomplished using the **`synchronized`** access specifier (monitor construct)

```
synchronized boolean withdraw(int amount) {
    ....
}
```

Or using the synchronized block:

```
boolean withdraw(int amount) {
    synchronized(this {
        ....
    }
}
```

Consumer-Producer…wait() & notify()

Scheduling

- How does the Java runtime schedules CPU time among threads?
  - If a thread is blocked (I/O) or sleep, it uses no CPU time
  - The runtime chooses the thread with the highest priority
  - If all threads have the same priority (then order is not defined)
    - Preempting threads (share processor)
    - Allow one thread to run till it gives up the processor. All other threads will be starved of CPU time
    - Because of this, you should not perform long compute-bound tasks without calling Thread.yield() periodically

HTTP Server

- It is an application-level protocol
- Supports several requests (e.g. GET, POST, HEAD)
- Examples of GET:
  - GET http://host.domain/doc.html HTTP/1.0
  - GET /doc.html HTTP/1.0

- Programming example: httpd.java

Security in HTTP

- How to protect against:
  - http://host.domain:port/../../../etc/passwd
  - http://host.domain:port//etc/passwd
- Implement a security manager