Multiple Clients

ComS 587X
Fall 2002

Server Designs

- Single Threaded
  - Single client
  - Multiple clients via select()/poll()
- Multiprocess
  - Multiple clients via fork() for each client
- Multi-threaded
  - Multiple clients via thread for each client

Single-Threaded Servers

- Single client
  - Easiest
    - Handle one client connection at a time
    - No resource management required
  - Poorest throughput
    - Each client must wait for all prior clients to complete

Single-Threaded Servers (2)

- Multiple simultaneous clients
  - Maintain state for each existing client
    - Reading request/processing request/sending response
  - Accept new clients
  - Juggle client connections via select/poll or asynchronous read/write/accept calls
  - No special language, library, or operating support required

Multiprocess Servers

- Multiple simultaneously-executing processes
  - One process to accept connections
  - One process per client
- Requires management of shared resources
  - Inter-Process Communication
    - Signals, locks, semaphores, shared memory, shared files, pipes
- Heavyweight context switching
  - Separate virtual memory spaces, file tables, etc.
  - O/S kernel involved every time a switch occurs between clients
- Doesn't scale well to thousands or millions of clients

Threads

- Traditional process execution
  - One thread (program counter and stack pointer) per process
  - Multitasking systems simulate multiple simultaneous processes by rapidly switching between processes
- Threaded process execution
  - More than one thread (PC and SP) per process
  - Multithreading processes simulate multiple simultaneous threads by switching between PC and SP sets

Examples of Multi-thread Programs

- GUIs
  - User-interface thread handling mouse & keyboard events
  - Computation thread(s) perform “real work”
- Web browsers
  - Multiple threads to handle simultaneous downloads of page elements

Threads Issues

- Thread scheduling
  - Priority
  - Pre-emptive or non-pre-emptive
- Shared access to common resources
  - E.g., variables/objects
  - Threads may execute in any order
  - **Atomic** operations
- Thread implementation
  - User level, or
  - Kernel level

Java Threads

- Two approaches:
  - Extend the java.lang.Thread **class**
    - Override the run() method to specify what tasks the thread will perform
    - Start the thread by creating a new instance of your class and invoking its **start()** method
  - Implement the java.lang.Runnable **interface** in your class
    - Provide a run() method that specifies the tasks to perform
    - Start the thread by using the Thread constructor on your class, then invoke the start() method on the new Thread instance

Interesting Thread Methods

- Constructors
  - Thread(),Thread(Runnable target), Thread(ThreadGroup group, Runnable target), Thread(String name)
- public void start()
- public static void currentThread()
- public static void yield()
- public static void sleep(long ms)
- public final void stop(), suspend(), resume()
- public void interrupt()
- public final void setPriority(int pri), public final int getPriority()
- public final void join()
- public final void setDaemon()

Thread Example 1

```
public class ExtendThreadDemo extends java.lang.Thread {
 int threadNumber;
 public ExtendThreadDemo ( int num ) {
  threadNumber = num; // Assign to member variable
 }
 public void run() {
  System.out.println ("I am thread number " + threadNumber);
  try {
   Thread.sleep(5000); // Sleep for 5 seconds
  } catch (InterruptedException ie) {}
  System.out.println (threadNumber + " is finished!");
 }
 public static void main(String args[]) {
 Thread t1 = new ExtendThreadDemo(1);
 Thread t2 = new ExtendThreadDemo(2);
 t1.start(); t2.start(); } }
```

Thread Example 2

```
public class RunnableThreadDemo implements
    java.lang.Runnable {
 public void run() {
  System.out.println ("I am an instance of Runnable");
 }
 public static void main(String args[]) {
  System.out.println ("Creating runnable object");
  Runnable run = new RunnableThreadDemo();
  System.out.println ("Creating first thread");
  Thread t1 = new Thread (run);
  System.out.println ("Creating second thread");
  Thread t2 = new Thread (run);
  System.out.println ("Starting both threads");
  t1.start(); t2.start();
 }
}
```

Summary

- Methods of handling multiple clients
- Single thread
- Multiple processes
- Multiple threads
- Java threads introduction