Today's Agenda

> Presentations

> Threading

> Synchronization

> Concurrent Data Structures

Threads

Threads are needed whenever your program needs to do two things at the same time.

Many frameworks already have some threading which may be hidden to the programmer:

GUI frameworks have input listeners which run on different threads to determine if there is mouse input or keyboard input, as well as a drawing thread.

OpenGL frameworks (GLUT, JOGL, etc) have a main animation thread which calls the display method as often as possible

Audio frameworks have a main audio thread which runs constantly to interact with the audio system

Threads

There are various reasons why you might want to spawn your own threads:

1) in general, whenever you need to more than one thing simultaneously

For instance, to listen to UDP packets on a port and to draw something (as we saw with the Datagram project)

2) when a particular process will take a long time and slow down or pause your main thread

For instance, an OpenGL game which executes code which evaluates the behavior of a NPC might take longer than the display rate and needs to be put in another thread else it will slow the rendering.

For instance, you might want to display a progress bar that shows how much of a file is being uploaded. One thread (the background thread) loads the file and the other thread checks the other thread to see how far along it is and displays that information accordingly

Threads

The main difficulty that arises when dealing with threads is when you have more than one thread which needs to manipulate the same data.

For instance, as a simple example, a program which draws shapes to the screen

Thread 1 : waits for user input via a GUI to decide what shapes to draw to the screen

Thread 2 : draws those shapes to the screen

The main data structure in this case might be a List of Shapes.

If Thread 2, which is drawing the shapes, accesses the List at the same time that Thread 1, which is updating the shapes, adds or deletes shapes, this can cause instability in your program.

In Java, where data structures are "fail fast", your program will exit with a runtime "Concurrent Modification Exception"

Synchronization

There are a few different strategies to deal with this situation:

1) have one thread make a complete copy of the data structure before it starts to use it

   For instance, the drawing thread could copy the data at the start of the display loop so that updates on the other thread do not effect it

If you have a lot of data this could be slow and memory intensive, and might trigger lots of garbage collection, etc.

2) create a "lock" of some kind (a mutex, a semaphore, etc) which checks a flag to see if it safe to access the data structure. These can be tricky to write sometimes, but there are built in methods in Java 6 and above

3) create synchronized methods to access your data structure. Synchronized methods automatically lock so that only one thread can access the method at a time. This will protect your data structure from being manipulated by multiple threads concurrently.

   This usually works well, but could potentially slow down performance if you often have lots of updates to your data structure

4) use a synchronized data structure and lock the data structure using the "synchronize" keyword before reading from or updating the data

5) use a concurrent data structure which automatically synchronizes the data structure

Code

code samples:

manual lock

synchronized blocks

synchronized lists

java.util.concurrent data structures