SortingInAction

Sorting in Action is a program which demonstrates simple use of Cthreads and Mach messaging. There are many unusual concerns about running multiple threads within the context of one application, most notably that the objects of the Application Kit were not designed to be re-entrant. This means if you try to send and receive messages to AppKit objects from multiple threads at the same time, you will end up with a mess on your hands. It is important to carefully construct your program to avoid such catastrophes. The Display PostScript system also does not take precautions for multi-threaded applications, so one should conduct all DPS interactions (that includes drawing, posting events ,etc) from only one thread--the "main" thread.

When a trial is started, I fork off a cthread to do the sorting for each algorithm. Since the thread cannot draw directly, it must ask the main thread to help it out each time that algorithm needs to draw. The main thread allocates a special port just to receive those drawing requests and monitors it by using DPSAddPort(). When a thread needs to draw, it creates a simple Mach message which contains the information about the drawing request. It sends the message using msg_send, and when the main thread receives it, a handler function is called which will have the appropriate SortView do the requested drawing. In this way, all the drawing is handled through the main thread.

You'll notice that the algorithms are compared on the basis of "ticks." A tick is an arbitrary

unit I created, so that the user has control over the expense of the different sorting operations. Counting ticks also allows me to synchronize the threads so no thread gets too far ahead of the others. If you're curious about how the Mach kernel schedules threads, try running a trial when you have set the value of each sorting operation to zero ticks. In this case, all my synchronization is disabled, and you can see how Mach tries to schedule the multiple threads. While Mach schedules, you'll notice that each thread has a good long run before it hands off the processor to the next thread. It makes sense that you don't want the kernel constantly context-switching, however, in my sorting program, I needed a finer granulity on the time-slicing. I accomplish this by maintaining a global variable TickCount which represents the elapsed ticks. When an algorithm takes the lead and has used more ticks that TickCount, it will update TickCount to its current ticks and put itself to sleep, waiting for another thread to catch up. When another thread passes it, the new thread in the lead goes to sleep and the previous sleeping thread is awakened and gets a change to catch up. In this way, no one thread gets to hog the processor for long. The sleeping and waking is accomplished by use of condition variables.

There a few global variables accessed by all the sorting threads which are protected by mutex variables. A mutex variable provides a "lock" for shared data which is used to ensure that only one thread at a time can access the global variable.

Classes of this application:

GenericSort This class encapsulates all the functionality for a sort object: managing the data, keeping track of tick counts, knowing how to animate the sort, controlling the speed, and more. GenericSort is designed to expect that the sorting will run in a thread, so its use the correct messaging for all drawing, rather than trying to draw directly. The class defines the methods like swap which will exchange the two data elements and send the proper message to the main thread for that drawing request. The specific subclasses then use the swap method and inherit all that messaging. As an abstract class, it lacks only a specific sorting algorithm. This makes way for a great inheritance--each of the specific sort classes only needs to implement a sort method.

BubbleSort
InsertionSort
MergeSort
QuickSort
SelectionSort
ShellSort These are the specific sort subclasses. There is very little to each of them, besides a sort method containing the particular sorting algorithm.

SortView This class is a custom view class which manages the display for a sort

object. It has methods to display the different steps in the animation (moves, swaps, compares, etc) which are called by the main thread when it receives a drawing request from a thread.

SortApp This application subclass handles the responsibilties of the main thread. It allocates and monitors the port for communication from the threads. The SortApp class also maintains a timed entry which updates the tick "clock" to show elapsed ticks while a trial is in progress.

SortController This class is in charge of running a sort trial. The SortController object is the target of the target-action methods of the controls on the SortParameters panel. These controls set the parameters for a sort such as data set size, which sorts to run, etc. When the SortController is asked to start a sort, it generates the data set and launches all the sort objects in their separate threads.

Other files:

Sort.nib The main nib file. Contains the Sort Parameters panel.
InfoPanel.nib The nib module containing the Info Panel (loaded only on demand).
HelpPanel.nib The nib module containing the Help Panel (loaded only on demand).
SortWraps.psw pswraps used by the SortView to animate the sorting.
IB.proj, These files created by Interface Builder.

Makefile,
SortingInAction_main.m,
SortingInAction.iconheader

Topics of interest from SortingInAction:

Forking and detaching cthreads (SortController)
Creating and sending a simple Mach message (GenericSort)
Allocating and monitoring a port using DPSAddPort (SortApp)
Thread synchronization using condition variables (GenericSort)
Protecting global shared memory with mutex variables (GenericSort)
Using a timed entry (SortApp)
Simple pswraps (SortWraps.psw, used by SortView)
Instance drawing (SortWraps.psw, used by SortView)
Using inheritance (GenericSort and all the sort subclasses)
Multiple nib files (Sort.nib, InfoPanel.nib, HelpPanel.nib)
Loading a nib module on demand (SortApp)
Entry validation using text delegate method textWillEnd: (SortController)
Using static class variables (GenericSort)
Storing and accessing strings using an NXStringTable (SortApp, SortView)