

Wait! That's not Reentrant!

2007/11/02

Sometimes when writing a program I will skip the design stage and just hack together a proof of concept for some feature to make sure I understand the problem. Once I have something that works, I go back, write up a real design, and rewrite that feature from scratch following a sane and extendable design.

My main project now is a little program to display useful information for people roasting coffee. How long has the roaster been on, how long has the current batch of coffee been in, what the current temperature from any number of thermocouples is, how the roast to the current point compares to a preset profile, things like that along with some data logging support. The goal is to reduce the need for paper, improve consistency, and make my job easier.

For temperature displays, I needed to learn how to use the interface to a device that reads the thermocouple channels and get a temperature to display or log or graph or whatever else I wanted to do with the data. I hacked together a little program that would loop continuously, calling a function to read temperature data and, if there was a new measurement available, display the temperature on the screen. This worked, but it was limited to reading only one thermocouple and it could only display that data. This was fine, but it was excessively resource intensive and not particularly reusable.

I decided that I really needed to split the program up into three different classes. A DAQ class represents the device that reads the thermocouples. These devices support multiple thermocouple channels, so a Channel class is used to represent each thermocouple. The DAQ class handles setting up the device, reading the data as it becomes available, and passing each measurement to the appropriate Channel instance. A TemperatureDisplay class is used to display the temperature data as it comes through the Channel. As many displays can connect to the channel as needed, and other sorts of objects can also be notified of new data from the channel. A model for data logging, for example, could be used instead of or in addition to the temperature display. None of these classes are tied strongly to any of the others, so I could write a new DAQ class to read data from other vendors or act as a simulator for testing purposes and everything else would still work without modification.

I wrote this, compiled, and it worked... between one and three measurements, at which point the program became unresponsive, `top` reported 103% CPU usage, and when I paused execution in the debugger, discovered more than 1300 stack frames. Stopping the program and stepping through the problem area in a debugger caused the problem to go away.

Today I investigated this and immediately saw the problem on the stack. I had assumed that the function called to get new measurements would always exit before it was called again. After all, I hadn't split off any threads and nothing the problematic function called would obviously call this function again. Yet, there it was, several calls to this function on the stack. It turns out that two of my assumptions were incorrect. The framework I used had 21 threads executing for my little single threaded application and code in the framework for passing data from the channels to anything connected to the channels could notice that enough time had passed to try to obtain another measurement. I put a lock around the offending function, and now that feature works perfectly. It seems that I will need to be a little more careful with functions that are not reentrant or thread safe.