Generalized Parallelization & Data Abstraction in C

Jordan Breen

Abstract:

This project explores the benefits of generalization and abstraction of data in program development. These two elements when used in tandem allow for quick scaling of projects along with code reliability. The development of more generalized forms of parallelization is the first step in parallelizing processes of serial software.

Introduction:

This uses a loading of data from a sqlite database to demonstrate how the task of parallelization can be broken down into steps that can be generalized and handle different forms of input. To further demonstrate this, all parallel operations are confined to files *thread.h* and *thread.c*. This model aims to be as separated from the operation and functionality of the other files.

Findings:

Parallelization in this model operatate with bag of task handling distribution, it simply hands out an auto-incrementing unique integer value within a supplied range. Generalization comes in with the void function pointers that function process_threads() requires. Wrapper functions visible to the user, via their declaration in thread.h, call their similarly named obscured and generic parallel process. This does three things, one limits the interaction the user has with parallelization, handles the thread creation, destruction, and error handling process in a contained way, and allows processes and functions to be executed in batches or groups. The last benefit mentioned is showcased in the thread.c obscured functions _init_resources and _free_resources. Both functions call to visible functions in resource.h to retrieve an array of function pointers otherwise invisible to anything other than calls in resource.c. These function pointers are reference to constructors and destructors found within each header file of object type structures found throughout the program. Each initialization function is responsible for statically loading table data from a single database file, parallel initialization becomes as easy as main calling init_resources. This limiting of access to the function implementations to only those given by the interface is a model referred to as Pimpl ("pointer to implementation"), opaque-pointers, or d-pointers, and is model I'd very much like to continue using in the future.

> "The Pimpl idiom is a simple yet robust technique to minimize coupling via the separation of interface and implementation and then implementation hiding. Successfully popularized by Sutter ([1,2]) under the cheeky Pimpl name (as for "pointer to implementation") the technique has been known long before under various names (the Cheshire Cat in [3], the Handle/Body in [4], the Bridge in [5]) and conceptually deployed in C and C++ as far back as at least early eighties ([6])." (Batov)

Conclusion:

I will most certainly be using this model for parallelization going forward, as it already has proven most useful in modifying serial code into parallel code. Data abstraction and encapsulation models in C has also proven itself to be incredibly useful in increasing the reliability and debugging speed of code developed in C.

References:

Chris McKillop. "Programming Tools — Opaque Pointers". QNX Software Systems.

Bruce Eckel (2000). "Chapter 5: Hiding the Implementation".

Vladimir Batov (2008-01-25). "Making Pimpl Easy". *Dr. Dobb's Journal*.