Thread Safe Graphics Library

Elizabeth Koning and Christiaan Hazlett
Dr. Joel Adams

Calvin COLLEGE 1876

Introduction

As single-threaded performance improvements have slowed over the past decade, multicore computers have become more and more common. The main advantage of this type of system is its ability to easily scale up. However, existing software must be rewritten to take advantage of the performance benefits.

Understanding the ways of synchronizing multiple processes and their shared resources has always been a tough step for students, but with the TSGL we can visually demonstrate the inner workings of these algorithms, significantly improving the learning experience.

Objectives

Our research goals this summer were to rewrite the library to support running the visualizations on the Raspberry Pi, as well as to create visualizations for three classic parallel problems:

- Producer/Consumer
- Reader/Writer
- Dining Philosophers

In addition to the above, we created a new object-oriented API for the library, and rewrote the existing demos using the new features.

The Classical Problems

Producer/Consumer

This problem demonstrates how multiple threads must be synchronized when there are some threads which produce a value and place it into temporary storage, while other threads remove the value from temporary storage and perform some action with it.

Potential issues include overflowing temporary storage or overwriting values. The programmer must ensure that the producing threads do not significantly outpace the consumer threads, or an imbalance of resources will occur.

Reader/Writer

In this problem, multiple threads request access to a shared data-store, with some threads attempting to modify the store, and some attempting to just read from it.

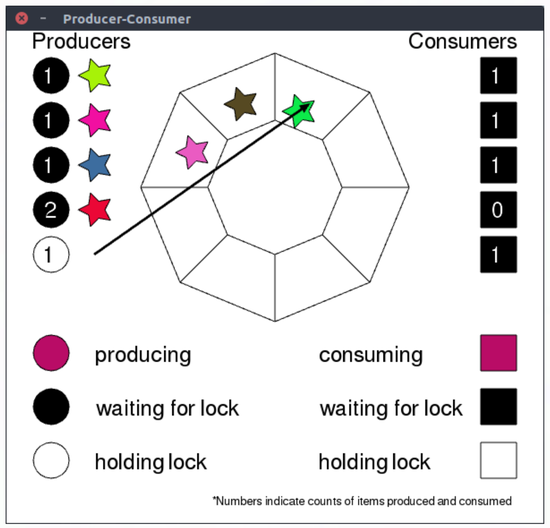
While multiple threads are allowed to read at a time, only one may write to avoid corruption of the store. Similarly, if one thread is writing to the store, the reader threads must wait for the writer to finish.

Dining Philosophers

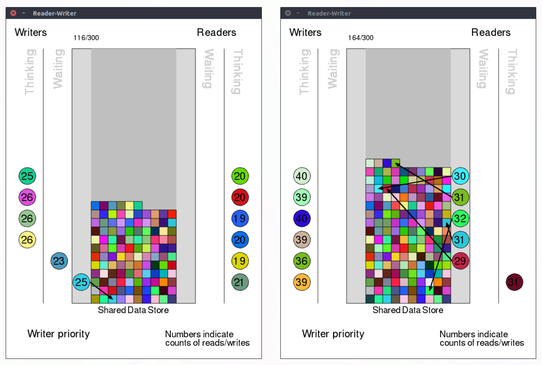
This problem demonstrates how threads must cooperate when they each request exclusive access to limited resources at the same time. The programmer must ensure that it is not possible for the threads to lock up, each waiting for another to release their locked resource.

Results

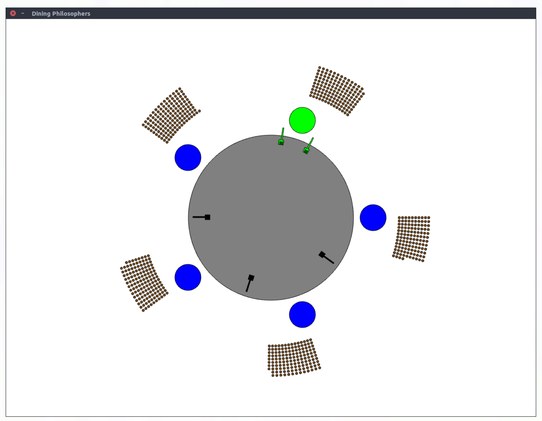
Producer/Consumer

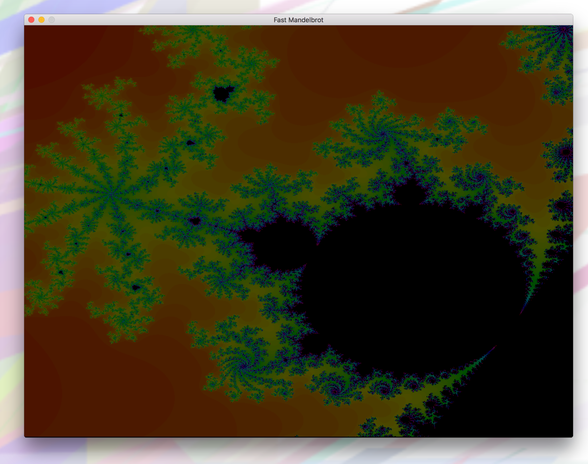


Reader/Writer



Dining Philosophers





The TSGL displaying a Mandelbrot fractal

Conclusions

By the end of the summer, we had successfully met our three goals: a working Raspberry Pi version, new visualizations, and an easier-to-use but more flexible way of interacting with the library.

This fall, we will hold a special event in the CS department to experimentally determine the effectiveness of these visualizations for teaching about multiprocessor synchronization.

Additionally, this spring multiple schools will begin using the new version of TSGL in their classes.

More Information

Full source code, examples, and references can be found online at our GitHub page: https://github.com/Calvin-CS/TSGL