CMPUT 379 Assignment 4 Report

By Nathan Klapstein #1449872

Table of Contents

Contents

Objectives

a4tasks

The main objective of a4tasks was to create a simulation of an extended dining philosopher's problem using C or C++ and multiple threads. In which, multiple non-sharable non-preemptable resources (i.e. chopsticks) must be shared between multiple tasks (i.e. philosophers).

Note: for a more detailed report on the objectives required by a4tasks please read:
https://eclass.srv.ualberta.ca/mod/url/view.php?id=3160968

Design Overview

a4tasks

- C++ was used over C due to its added functionality
- Class-based structure was used to separate functionality
- Major functionality/classes were placed in separate files for maintainability/readability
- Used a cmake file over a makefile for development (more modular/futureproof)
- Majority of print statements have starting strings of ERROR: WARNING: INFO: and DEBUG: for ease of debugging
- Simple header only file was used for defining the TASK structure task.h
- TASK structs content was ordered for potential optimizations (smallest data to biggest data)
- Travis-CI was used to implement continuous integration (CI)
- cpplint was used to validate the styling of source code
- Multiple test files (located in the `test/` directory) are automatically executed within Travis-CI for a simple testing/CI cycle
- Simple simulation start/entry point was defined by the `start()` method.
- a4tasks.cpp handled argument parsing and called simulation.cpp's start() method to initialize the simulation
- pthreads was used to initialize TASK threads for the simulation
- common.cpp included functionality common between files (such and wrappers for locking and unlocking mutexes)
- mutexes were used to ensure proper concurrent memory usage

Project Status

a4tasks

Status: Completed – Automated Testing

Some difficulties encountered in the design of a4tasks was in understanding the threading and mutex locking API given within C++ and POSIX. It is old, largely unchanged, and very rough around the edges. Doing multithreading in C and C++ is no easy task as both languages lack deep compile-time checks, and lack effective error reporting/bubbling. This led to manual debugging being the only solution for remedying errors within a4tasks, which resulted in much time being wasted. Furthermore, C++ allows for nasty operations by default (especially in threading) that can lead to memory management issues and

thread operation issues. C and C++ place the correctness of the program mainly in the hand of the programmer, which, is the most likely competent fail. Due to this, much time was spent remedying self-onset failures.

Modern languages such as Rust avoid some of these pitfalls by incorporating strict rules (i.e. “safe” Rust) and placing emphasis on non-sharing, non-pooled structures which are both error-prone and insecure. Rust places emphasis on the idea of passed messages instead of pooled data. Usually, this leads to easier to understand procedures and much safer execution.

Testing and Results

a4tasks

Testing for a4tasks consisted of simple runtime testing over a set of test files contained within the `test/` directories. These tests are executed via Travis-CI on each push (which immensely helped in quickly identifying issues on changes). Instead of rewriting the test’s output I would recommend viewing test results directly at https://Travis-CI.com/nklapste/a4tasks. Also viewing each test file within `test/` also provides comments on what each test attempts to validate.

Some tests include validation of:

- standard operation with a 5 unique single resource 5 task problem
- operation with tasks with varying resource arguments
- operation with an input file with no specified tasks
- operation with resources and tasks that would lead to a non-finishing system

Sources

a4tasks Code Source: https://github.com/nklapste/a4tasks

a4tasks Travis-CI Builds: https://travis-ci.com/nklapste/a4tasks

Acknowledgments

cplusplus.com for C++ documentation:

http://www.cplusplus.com/

C++ undefined behavior references:

https://en.cppreference.com/w/cpp/language/ub

CLion IDE for C++ refactoring and development tools:

https://www.jetbrains.com/clion/

Tutorial on Cmake:

https://www.cs.swarthmore.edu/~adanner/tips/cmake.php

Travis-CI C++ templates:

https://docs.Travis-CI.com/user/languages/cpp/