Shawn Rast, Jinxian Zhu, Tien Anh Nguyen, Hao Lin

Methods

Clock:

The clock method is a fairly simple one. The main method only makes a new thread. That thread then pauses execution for a second, runs the 'clear' command (which clears the terminal). Then, the current time is stored in a variable, and then ctime converts it to a string and prints it.

CriticalSection:

This method's goal is to allow two threads to take turns working, despite random wait times for each thread. Each thread is given a flag, used to mark whether it has started; an indicator of which thread it is (either the first or the second thread); and the other thread's flag to check whether the other thread has finished.

This is the flow: The first thread sets its flag to true, indicating that it is working. Then, it checks to see if the other thread has started and if it is the other thread's turn – if not, which it isn't because the thread started first, states that it's entering its critical section, waits a random amount of time, and prints it's leaving the critical section, and then sets its flag to false, indicating that it has finished.

The second thread cannot begin printing until two conditions are met: it must be the second thread's turn, and the first thread has to be finished. It simply waits until that is true, and then indicates to the other thread that it is finished. This in turn allows the other thread to escape its while loop, and so on. It means that the threads will always be interweaved, with each thread taking a turn.

Synq:

This method consists of two threads. One adds a rhyme, line by line, into a shared queue. The second constantly checks that queue, and prints out the results when the entire rhyme is present.

The second method only knows that the first method is done when the first method puts the word “done” in the queue they share. Because they share the same queue, the second thread can continually check the end of the queue it has to see if the first has finished.

Sum:

The sum method is very simple. The main function inputs a number, and sends that number to a thread. That thread then calculates sum of the integers from 1 to the number by looping, and prints it to the screen. The main thread then joins the spawned thread and closes.

ProducerConsumer:

This method has two threads, and one shared queue between them. The queue stores time_t values. The first thread (producer) pushes time objects into the queue, and the second pulls them off if they exist (consumer), each after waiting for a specified amount of time. Because both threads share an object, they can operate independently.

Overview

Multi-threading is, in short, splitting up a program into multiple smaller sections that run simultaneously. They can have variables that are shared between them, as well as variables that are unique to each thread. Because they’re part of a larger program, their results usually need to be processed by the original thread that made them – and this requires synchronization to work properly. Using shared variables to mark when a thread is running, halting threads in certain conditions, and using thread.join() allows the original process to manage the other two processes in order to effectively divide tasks.