CMPUT379 Assignment 4

Chengxuan Li

April, 7th, 2022

1 Objectives

This assignment is to write a C/C++ program that utilizes pthreads to simulate the concurrent execution of a set of jobs. The jobs occupy a resources pool which might have different number of resource types, and each type of resource might have different number of units.

2 Design Overview

1. There's one resource pool in the entire simulator. Each type of resource will have one corresponding mutex to control access and modification. When a thread wants to get the information about one type of resource, or want to allocate or release one type of resource, it needs to first grant the mutex before accessing and modification.
2. Each thread needs to hold all needed resources before executing its assigned job. During the resource allocation of a thread, it can hold (preempt) the resources that were already allocated. If a thread detects that it cannot allocate the needed resources to execute its assigned job, it will immediately releases all hold (preempt) resources back to the resources pool; then, it will wait 10 ms and try again.
3. There's map to record status of each job and other related information. Each thread has its own mutex for changing status so that it does not interfere other process when it need to change its status. When the monitor needs to print out the status of all threads, it will first collects all the mutex for changing status from all threads such that all thread cannot change its status during the display process. The monitor will release all those mutex back afterward. Notice that the monitor will start running before the main thread starts reading the input file and spawning any job threads.

3 Project Status

All requirement in the assignment spec are implemented and tested. First issue is that lower monitor time will block all job threads more frequently because it prevents all job threads to change its status. All job threads need to wait for changing their status before continuing their jobs. Second issue is that some threads will have higher wait time than other because of the deadlock prevention being used (all or nothing). All or nothing strategy can cause some degree of starvation during the simulation with very limited shared resources (dining philosophisers problem), where one thread might wait more than other threads. Lastly, there's a problem occurs when multiple threads (either job threads or monitor thread) try to display message using "cout". The messages from different threads can collide / merge together, which is might because "cout" is not thread safe. It can be fixed by adding addition mutex for outputting messages. Each thread need to acquire the mutex before printing a message. However, it isn't implemented in the simulator because it will further slow down the execution time and reduce concurrency.

4 Testing and Results

The output format of the message matches with the one specified in the assignment spec. For Dining Philosophers Problem with 5 people, the wait time of each thread were around 200 ms to 1000 ms, and the running time of the entire simulation was around 4000   4900 ms on. The number of iteration was 20, the monitor time was 250 ms, and the number of cores was either 3 or 6. This result was obtained from my local machine whose OS is Archlinux. On lab machine, the lower and upper bound of the wait time and running time is slightly higher.

5 Acknowledgements

List of sources of assistance: None