Programming assignment 1

Getting started

All your submissions should be implemented in Go unless the problem specifies something different. You can download Go at https://go.dev/dl/. You can also use various package managers to install it, e.g., HomeBrew on macOS.

Problem 1

Implement a binary heap that multiple goroutines can safely use. It is ok to use coarse-grained locking, i.e., lock the heap in each method that accesses it.

Problem 2

Implement a deque using a linked list that multiple goroutines can safely use. Your locking strategy should be as fine-grained as possible (but you must use mutex locking). Describe the strategy in the report and argue for why it is minimal.

Problem 3

You know that the string `a74277500228f7b4cfa8694098443fc5` is an md5 hash of a password. You also know that the password is six characters; the only characters allowed are `a-z0-9`. Your task is to create a parallel password cracker in Go.

You can use `crypto/md5` to create an md5-hash of a string. You have cracked the password when you find a string that generates the same md5-hash as the one above.

Your solution should use goroutines and channels. Describe your design in the report as well as benchmarks of how well it scales with an increasing number of goroutines.

Problem 4 (required for C+)

A mutex lock or a binary semaphore can be used to allow a single thread or goroutine to access a critical region. A counting semaphore can allow up to N threads or goroutines to access a critical region. Implement a counting semaphore using mutex locks. N should be a parameter. Describe your design in the report and provide a simple example showing it works.

Problem 5 (required for C+)

Modify the heap in problem 1 to use a reader/writer lock (`RWMutex`). The methods should be modified to use the correct type of lock/unlock operations.

Submission guidelines

Submit your solutions as a single zip file via Moodle no later than 17:00 on February 16, 2024 (cutoff 08:00 February 19). This is a group assignment that can be done in groups of one or two students. Your submission should contain well-structured and organized Go code for the problems with a README.txt (or .md) file describing how to compile and run the Go programs and a PDF report describing your experiment and findings from Problem 4.