Homework 2, Part 3

Cooper Faber

May 2021

1 Experiments

AVG OF	COARSE	RW	SWAPTOP
POPS	402368	88420	26909
PEEKS	362968	1300350	1381000
PUSHES	266792	6524	3999.5
SWAPS	N/A	N/A	7627

1.1 Variation and Analysis

Although the actual numbers varied quite a bit, likely due to system load, the ratios for coarse and r_w stayed similar throughout the runs. Coarse just locks every function while it's being used, and r/w has a hard lock on push(), a read lock on peek(), and a read lock for the check on pop() before a hard lock is engaged, in case the read check fails (i.e, the stack is non-empty). This lends itself to minimal variation in ratios, as the r/w lock heavily favors the peek operation. The SwapTop stack, however, was highly variable in pops and pushes - I believe this is because of the read/write check inside of the SwapTop function. If the value on top of the stack is the same as the value attempting to be swapped, SwapTop will not do anything, and will only read. This will allow the peeks to continually go, and have a high share of the throughput, blocking pop() and push(). However, if SwapTop is writing consistently, the ratios will skew more towards pop() and push(), although peeks will still be a large majority due to the nature of the locks.

1.2 Swaptop Solution

My solution to SwapTop was hinted at in a piazza post. It involves using the read lock to check if a write is actually necessary, before locking the stack, iterating through, and replacing the data. Given that there are only three valid data options, a write will only be necessary about 66% of the time, so the other 33% can be optimized to allow peek() to happen as well, hence the large peek() numbers. Ironically, this optimization allowed me to go back to r/w and implement that on pop() as well, as that function also has the unique characteristic of occasionally returning without writing (albeit much less often).