Andy Mahoney

0733921

Write-Up

For locking and preventing deadlock the same scheme was moved for row or cell, in the following description the words can be interchanged. For locking the cell/row enters a loop, in the loop we first lock the first cell/row then we enter a conditional. The condition is if trylock on the second cell/row returns as a success. If so we lock the second cell/row and break out of the loop, where the code we protected lies. If trylock fails we unlock the first cell/row and a repeat the loop. We do this so that if one thread is struggling to lock its second cell it can release the first one giving another thread who may be waiting on that first one a chance to lock it and complete its swap. This prevents the dining philosophers problem. After the swap is completed then the locked cells/rows of that thread are unlocked. We also consider the possibility that cell1/row1 and cell2/row2 are the same cell/row in which case we only rock once and unlock once. Also to prevent data corruption threads\_left is locked before being edited and unlocked after. For locking the grid deadlock cannot occur since only one thread, who is doing changes can have it so we just lock the grid, do a swap, then unlock it.

The graphs in data-Spreadsheet.xlsx Sheet 1 are of the average and the variance. The average shows us that none is the fastest, followed by cell, then row, and finally grid. This graph is measured in seconds on the y-axis. The variance shows that none and grid had practically zero variance while cell and row had a lot a variance, particularly when the grid size was small. This is because cell and row had to deal with locking where a thread could be using the resource another thread wanted causing certain threads to have to wait before completing their operation, this overlap became more frequent as the number of possible cells/rows to use became smaller. Meanwhile none could just execute with out having to lock or worry about other threads, and in grid every thread had to wait in a line to be able to use the grid which played out in nearly identical ways every time. This is why grid size difference only impacted cell and row, and not none or grid. However while grid sized negatively effected smaller granularity, sleep(1) effected larger ones. Larger granularity meant less threads could work on the data at once so sleep(1) was added onto the total run time more often, since in cell multiple threads at once could do their sleep(1) at the same time but grid meant every thread had to run sleep(1) one at a time, one after the other.