Dylan Tymkiw

Project 3: Red Black Tree

Abstract:

The goal of this project is to implement a data structure that consists of a self balanced binary tree. Said tree is called a red black tree and I implemented it in python. This tree is intended for use with a completely fair scheduler.

The Node Class:

The Node class holds the key, which is the vruntime of the process, the data, which is the process itself, parent, left and right children, and a boolean telling us if it is Red or not.

I also implemented setters and getters.

The Tree Class:

insert():

This method takes the vruntime of a process and the process itself and inserts it to the tree. When the tree is empty, the given node becomes the root, if not, a helper method fix_insert() is called to balance the binary tree.

fix_inset():

Fix insert takes a node and sees if it is correctly positioned in the tree, if it is not, it balances the tree out by calling the appropriate helper methods.

print_tree():

This method traverses the tree and prints the nodes in order.

Extensions:

remove_min_vruntime():

This method removes the left most node in the tree which also is the min vruntime. It then makes sure that the pointer points to the correct node.

completely_fair_scheduler():

I attempted to implement the completely fair scheduler but I ran into problems when testing it since my tree loses some balance when a node is removed and then the root or a node in the right ends up being the min vruntime which causes the scheduler to crash or just not work properly.

*A jupyter notebook of me attempting to test the scheduler is included in my project.

Discussion:

This project was especially challenging since we had to implement a data structure we did not know much about and then attempt to use it for a scheduler. I learned a lot about red black trees but unfortunately not a lot about the scheduler since I could not get it to run properly.

Acknowledgements:

Naser Al Madi.