COP 5536

ADVANCED DATA STRUCTURES

PROJECT REPORT

Submitted to :
Dr. Sartaj Sahni

Submitted By:
Aditya Dutt
UFID : **14530933**
aditya.dutt@ufl.edu

Aim: To implement a simulation of job scheduler for an operating system using Red Black Tree and min heap.

Job Scheduler:

The code consists of a min heap in which jobs are stored with execution time as their key. The jobs are simultaneously inserted into red black tree also with Job ID as key. The job with minimum executed time is dispatched for minimum of 5ms and remaining time to finish job. In case the job does not complete in 5ms it becomes a candidate for the next scheduling round and the executed time is updated in both min heap and Red Black Tree. If the job will complete during this processing it is removed from both data structures.

I have used C++ language for this project.

Function Prototypes and Structure:

Min heap functions:

- swap (position 1, position 2) – It swap position of 2 elements in min heap during heapifying.
- find_left_child (parent) – Finds left child of parent.
- find_right_child (parent) – Finds right child of parent.
- rearrange(condition) – This function is used to heapify. If the condition is 1, then it bubbles up the element in heap. If the condition is 0, then it bubbles down the element in heap.
- find_parent (child) – Finds parent of given element child.
- insert_val(Template object1, Template object2) – It inserts (total job time, job ID) in heap.
- remove() – It removes an element from heap.
- get_size() – Returns size of min heap.
- set_size(int) – It sets the size of min heap to passed value.
- last_notnull() – Gives the address of last element in heap.
- dispatch1() – Dispatches a job for processing.
- display() – Displays the elements of min heap .

Red Black Node functions:

- Recolor() – It recolors a node to balance the tree.
- GetJobId() – Returns Job ID of a node.
- GetJobTime() – Returns Job time of a node.
- find_left_child() – Finds left child of node.
- find_right_child() – Finds left child of node.
- Clear_Parent() – Deletes parent of a node.
- Set_left(node1) – Sets the node1 as left child of a given node.
- Set_right(node) – Sets the node1 as left child of a given node.

Red Black Tree functions:

- AccessNode(root , jobId , time) – Accesses a node and updates the time of job with Id – jobId.
- AccessNode1(root , jobId) – Accesses a node with Id jobId and returns it's executed and total time. If no job with Id - jobId exists, it returns -1.
- Print_inorder1(root , jobId1, jobId2) – Gives range of nodes between jobId1 and jobId2.
- Path(root, vector , jobId) – Returns array of elements used for NextJob() and PreviousJob().
- insert_node(root, jobId, t)- Inserts a node in RB tree with Id – jobId and time- t.
- delete_node(root , jobId) – Deletes a node with ID- jobId.
- RR fix(root), Left_childRotate(root), Right_childRotate(root), height_changereduce(root), L_rotate_1(root), L_rotate_2(root), L_rotate_3(root), L_rotate_4(root), R_rotate_1(root), R_rotate_2(root), R_rotate_3(root), R_rotate_4(root)- These functions are used to rebalance the tree by left rotation, right rotation and color changing.
- AccessNode(ID, time)- This function is called from main function to access a given ID and update it's time. This function further calls AccessNode(root, jobId, time) as it is a private function.
- AccessNode2(jobId)- This function accesses AccessNode1 to return executed and total time corresponding to a given jobId.
- Next_or_Prev(jobId, int f) – Executes NextJob() if f=1 and PreviousJob() if f=0.
- level_traverse() – Prints RB tree according to level traversal.

As the program starts, it reads the input file. A global counter is set. When the arrival time of a job matches with global counter, the job is inserted.

The function insert_val(jobId, time) inserts job in min heap and the function insert_node(jobId,time) inserts job in RB tree.
Then the function dispatch1() takes the job with minimum executed time and processes it.
When the job is processed after each millisecond, the value of executed time is updated in RB tree using AccessNode(id, new_time) function. If the job finishes during processing, remove() function deletes the node from min heap and delete_node(jobid, time) deletes that job from RB tree.
After each processing, rearrange() function is called to keep the min heap balanced.

If the new job is PrintJob(jobId1) then function AccessNode(jobId, time) is called.
If the new job is PrintJob(jobid1,jobid2) then Print_inorder1(root , jobId1, jobId2) is called. It finds the elements in that range by moving to a subtree of a node only if the node is contained in the range as it is a balanced search tree. If the job is NextJob(jobid) or PreviousJob(jobid) then Path(root, vector, jobId) is called. If a previous or next job exists then it writes that job in output_file.txt otherwise it writes (0,0,0).

When the file is completely read, program only executes function dispatch1() because there is no new job to arrive.

At all places, vectors are used to store temporary values. Because vectors offer a lot more flexibility than array in terms of memory usage.

Time Complexity :

Min heap - The heap has ceil(log2(N+1)) levels. The insert swaps at most once per level, so the order of complexity of insert is O(log N). Since the remove swaps are at most once per level, the order of complexity of remove is also O(log N).

Red Black tree -All RB tree queries are executed in O(log(n)+S) where S is the nodes visited.

Tools and Technologies used:

Programming language: **C++**
IDE : **CodeBlocks**