Advanced Data Structure
Programming Project
Job Scheduler

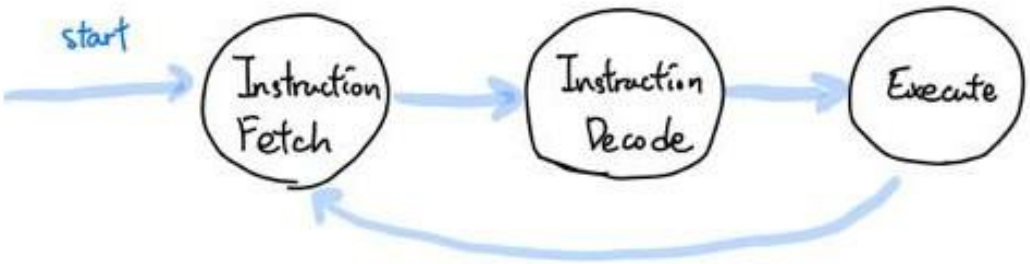
Chia-Hsing, Chu

UFID: 97316559

chiahsinchu@ufl.edu

Abstract

In this project, I use the min-heap and the Red Black Tree to store the executed time and the JobID respectively. And the purpose of this project is like understanding how CPU working in the computer, the time appear is like the address, and the min-heap and Red Black Tree is like the memory, and the commands is like the instructions. For example, the CPU will first fetch the instruction, then decode instruction, and then execute the instruction, keep looping as below.



In Scheduler, void PrintJob(string operand) which is used to print out one job or several jobs in a range. I use a find() for searching the comma(,) , so I just need to use one function with low and high to determine one job or multiple jobs in a range search. And I created a instruction set in opTable which listing what our system can do, and also a function pointer points out how to execute instruction. Then in run(), at beginning we initialized the busy to false, so in the for loop of run(), it will enter the (busy = = false) first, then we choose the smallest executed_time job, assign appropriate left_time and change busy to true for it to execute. After this, there's one job is executing, we need to update its executed_time and left_time. Also when executed_time = total_time, meaning that the job is completed, we have to remove it from both data structures. If it used up all the left_time, meaning that the time slice is out of use, but the job is not completed. Then we have to update the heap structure because the key of that job has changed. And the nextCommand() which is used to check if the appear_time of the command is same as the global time.

Function Prototypes

In this project, I use 1 header file (common.h), and 4 cpp files (heap.cpp, tree.cpp, scheduler.cpp, and main.cpp) to implement. In common.h file, I've included several structures with its parameters, functions, and variables that I will actual use in the implementation cpp file.

- Struct TNode
  Have variables such as jobID, totalTime, color, link, left, right to represent node in the RedBlackTree and link to min-Heap.

- Struct HNode
  Have execueTime, and the link to RBTree

- Struct Tree
  Have variables like root, size…
  Functions as below:
  TNode insert(int jobID, int totalTime):
  Insert the command at right place in to both tree and heap.

  TNode *search(int jobID):
  Search the particular key if it's in the tree or not.

  TNode *successor(int jobID):
  Find the next node.

  TNode *predecessor(int jobID):
  Finds the previous node.

  void remove(int jobID):
  Remove particular node in the tree.

  TNode *minimum(TNode *r):
  Find the minimum of the right substree.

  TNode *maxmum(TNode *r):
  Find the maximum of the left subtree.

  void leftRotate(TNode *x):
  Depends on different cases (such as color, sibling…), we need to do the right or left rotate to not violate the rule of the RBTree.

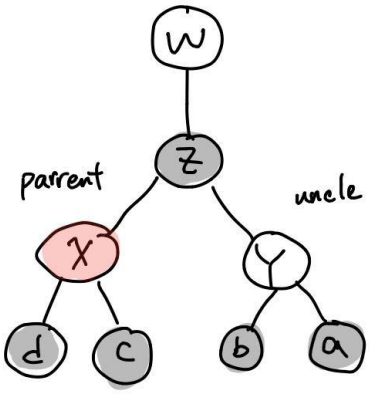
  void rightRotate(TNode *x):
  Same us top.

  void insertFixup(TNode *node):

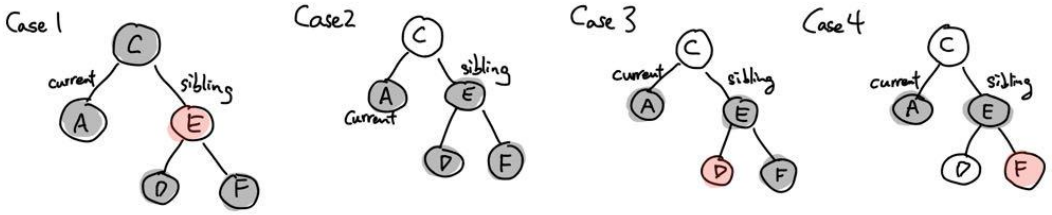
When two consecutive red nodes connected, need to adjust the tree.

void removeFixup(TNode *node, TNode *parent):
Depends on different cases (such as what node are you deleted, what color, violate the rule to adjust the tree.

insertFixup
There are 3 cases when we dealing the insertfixup problem:
Case 1: When uncle is red, no matter the insert node is the left or right child of node X.
Case 2: When uncle is black, and the insert node is the right child of the node X.
Case 3: When uncle is black, and the insert node is the left child of the node X.



removeFixup
There are 4 cases when we dealing the insertfixup problem:
Case 1: When sibling is red.
Case 2: When sibling is black, and both children of the sibling is black.
Case 3: When sibling is black, the right child of the sibling is black and the left is red.
Case 4: When sibling is black, and the right child of the sibling is red.



- Struct Heap
  It has vector for storage command data, and variable size.
  Functions as below:

  HNode* insert(int execTime, TNode *l):
  Insert new Job node into the heap.

HNode* getMin():
Get the job at root.

void deleteMin():
Delete the job at root.

void siftDown():
Compare if father is need to swap by the child, if so do the adjustment.
bool empty() {return h.empty();}:
Determine if the heap contain no nodes or not.

- Struct Command
  To store each read job with time_appear and commands.

- Struct Scheduler
  Have variables and functions as below:
  vector<Command> commands;
  unsigned eip: ←Next command to execute.
  int sys_time: ←System time counter
  HNode *current: ←Executing currently job
  bool busy: ←System state
  int left_time; ←Time slice (ex: less than 5 = time; greater than 5 = 5)

  Scheduler(char *filename);
  To read the commands.

  void run():
  Run our system.

  Void nextCommand():
  To distinquish what next command is.

Results

The platform for executing is on UBUNTU 16.04
make
./jobscheduler sample_input3.txt

Output file screenshop: