ITU Computer Engineering Department
BLG 335E Analysis of Algorithms I, Fall 2022
Project #3
Due December 28, 2022 11:59pm

Problem Definition

The scheduler is a component of the operating system of our computers. As its name suggests, it schedules, or alternates the resources provided to processes. The scheduler gives permission and resources for a particular process to run. Then, depending on the structure of the underlying algorithm, it may preempt, or take away that resource and provide it to another process. One of the most dangerous things that can happen when scheduling multiple processes to take turns using particular CPU resources is deadlock. A deadlock occurs when processes are holding onto resources such that there are not enough resources for any of them to continue. Each process waits for the others to finish, causing an issue called starvation.

In the literature, there are many algorithms that are designed to prevent deadlock. One particular algorithm is the Completely Fair Scheduler (CFS). CFS keeps a virtual running time (time allotted for a process to date) of each process and increments this running time by the amount of time that process was given CPU privileges. The run time for a process can be expressed as a monotonically increasing running sum:

$$VirtualRunningTime = VirtualRunningTime + AllocatedTime$$

When a process arrives, its initial virtual run time is calculated. Inputs such as a nice value can act as a multiplier for a particular process to receive more or less CPU time–depending on its priority. At each scheduling point, newly arriving processes are added to a Red Black Tree. In order to choose a process to run, CFS selects the process with the minimum virtual running time. The virtual running time of the process is updated, and if the process has not run to completion, then it is inserted back into the tree to await its turn. At the next scheduling point, the process with the minimum run time is selected again. Because the minimum run times are incremented, the upcoming orders of processes are shuffled. Therefore, deadlocks and starvation are avoided. For more information on CFS, check out the link provided[1].

A description of the expected implementation can be given as follows:

1. Processes will be read according to their arrival time and stored in an RBTree structure (RBTree rules apply.) according to their vruntime. A special case is that processes may have the same vruntime. In this case, the process that is read/arrived later is added to the right child of the current process.
2. Check for incoming processes. Add any to your RB-Tree that have an arrival time less than or equal to the current CPU time.
3. If there is a running task, and its vruntime is greater than the minvruntime, then remove that and add it back to the tree with its updated vruntime value.
4. If there is no task running currently, choose the task with the smallest vruntime on the tree. Remove that task from the tree to give it CPU privileges.
5. Update the currently running task by incrementing its vruntime value. Check whether or not it should stop running.
6. Repeat until the max allotted time for the simulation is finished, or all the processes have come to completion.

Important Note: Any addition or deletion to/from the red-black tree should be followed by the appropriate ***fixup*** routine.

[1] https://docs.kernel.org/scheduler/sched-design-CFS.html

Constraints

- The arrival time and run time required inputs are not timestamp values but rather integer value. You are free to simulate time in an abstract form (simply run through with a loop)!
- Please make sure to implement all of your tree methods naively. That is, your code should be written by you and you only.
- No use of STL is permitted.

Sample Cases

Your code should read the processes from a *.txt* file. The desired format for the input is given in Listing 1. The output format should be as provided in Listing 2.

Code Listing 1: Input File Format

```
NumProcesses SimulatorRunTime
ProcessID TimeOfArrival BurstTime
```

Code Listing 2: Output File Format

```
CurrTime, RunningTask, TaskVruntime, MinVruntime, RBTTraversal, TaskStatus

Scheduling finished in ... ms.
x of y processes are completed.
The order of completion of the tasks: x-y-z
```

*Note that the RBTree status will be provided using **inorder traversal**.

An example input file is given in Fig. 1, and the related simulation is illustrated in Fig. 2. *Your code is expected to work correctly in any input scenario.* Further examples for input and outputs are provided in Listings 3-6.

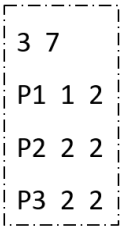

Figure 1: Sample of processes file

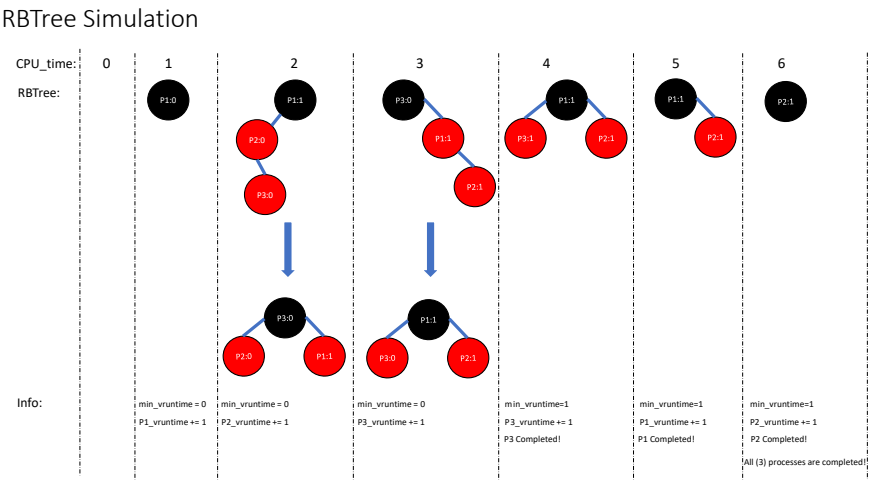

Figure 2: Simulation of RBTree-based CFS

Code Listing 3: Example Input-1

```
3 7
P1 1 2
P2 2 2
P3 2 2
```

Code Listing 4: Example Output-1

```
0,-,-,-,-,-,-,-
1,P1,0,0,P1:0-Black,Incomplete
2,P2,0,0,P2:0-Red;P3:0-Black;P1:1-Red,Incomplete
3,P3,0,0,P3:0-Red;P1:1-Black;P2:1-Red,Incomplete
4,P3,1,1,P3:1-Red;P1:1-Black;P2:1-Red,Complete
5,P1,1,1,P1:1-Black;P2:1-Red,Complete
6,P2,1,1,P2:1-Black,Completed

Scheduling finished in ...ms.
3 of 3 processes are completed.
The order of completion of the tasks: P3-P1-P2
```

Code Listing 5: Example Input-2

```
3 5
P1 1 2
P2 2 2
P3 2 2
```

Code Listing 6: Example Output-2

```
0,-,-,-,-,-,-,-
1,P1,0,0,P1:0-Black,Incomplete
2,P2,0,0,P2:0-Red;P3:0-Black;P1:1-Red,Incomplete
3,P3,0,0,P3:0-Red;P1:1-Black;P2:1-Red,Incomplete
4,P3,1,1,P3:1-Red;P1:1-Black;P2:1-Red,Complete

Scheduling finished in ...ms.
1 of 3 processes are completed.
The order of completion of the tasks: P3
```

Deliverables

- Your efficient C++ Implementation of CFS using RB Trees. [70 pts]
  - Please write your code clearly.
  - Provide detailed comments.
- Report detailing your implementation. [30 pts]
  - We expect your report to be in your own words.
  - We expect that you write clearly and proofread your work before submitting.

Report Structure

Please answer the following questions in your report. Make sure to use your own words. Write your report in sections, and provide visuals when necessary.

- **Section 1: Description of Code** Explain briefly the data structures you've constructed. Discuss the attributes and methods you've implemented. Provide pseudocodes of the methods in your implementation.
- **Section 2: Complexity Analysis** What is the piece-wise AND overall complexity of your implementation? Provide a detailed discussion and upper bounds.
- **Section 3: Food For Thought** Answer the following questions briefly.
  - Can you think of any advantages of using the RB Tree as the underlying data structure?
  - Is the CFS used anywhere in the real world?
  - What is the maximum height of the RBTree with N processes? Upper bound T(N)=? Prove it.

Submission Rules

- You should write your code in C++ language and try to follow an **object-oriented methodology** with well-chosen variables, methods, and class names and comments where necessary.
- You **cannot** use the C++ Standard Template Library (STL) algorithms. In addition, the use of specialized STL containers like deque, queue or priority queue is not permitted, but the use of vector or list is allowed.
- You can define multiple classes in a single cpp file or use multiple cpp files with header files.
- It is mandatory to include a MakeFile which makes your code compiled with the "**make all**" command.
- Also, make sure that your code can be run **on our Docker container** in the form of **./homework3 [input_file]**.
- If the code is not self-explanatory and does not include adequate comments, a point penalty of up to 20 points will be applied.

  All the reports must be prepared in LaTeX platform. You can use the following template: Sample template.
- Hand-written reports will not be accepted. Also for pseudocodes, please prepare them properly, do not copy and paste your exact C++ code that you used for your assignments.
- Do not share any code or text that can be submitted as a part of an assignment (discussing ideas is okay).
- Only electronic submissions through Ninova will be accepted no later than the deadline.
- You may discuss the problems at an abstract level with your classmates, but you should not **share or copy code** from your classmates or the Internet. You should submit your **own individual** homework.
- Academic dishonesty, including cheating, plagiarism, and direct copying, is unacceptable.
- If you have any questions about the homework, you can send an e-mail to korkmazmer@itu.edu.tr and ayvaz18@itu.edu.tr
- **Note that **YOUR CODES WILL BE CHECKED WITH THE PLAGIARISM TOOLS!**