Project (Part 1)

Weight: 10%

DUE DATE AND TIME: 10:30 am, 31st May, 2021

Requirements

- Write the program using C++
- Use the array-based implementation of heap discussed in the class
- Do not use any built-in function other than measuring the system clock time

In this problem, you are required to implement a min-heap (priority queue) to schedule real-time processes. The heap should be stored in a fixed-size array of 1,000 records. You must provide functions for inserting or extracting an element in/from the heap.

Every time you insert or extract an element from the heap, you must print:

- The number of levels in the tree traversed to re-heapify (adjust) the tree, as well as
- The time taken for the entire operation.

The difference may be zero if your system clock does not update the time quickly enough, but otherwise should accurately report how many nanoseconds (billionths of a second) it took to execute your code.

Your priority queue uses the heap to provide following functions of the priority queue:

- *insert_with_priority*: add an element to the queue with an associated priority.
- *pull_highest_priority_element*: remove the element from the queue that has the *highest priority*, and return it.

Finally, you must provide a driver program which reads a file and adds or removes things from the priority queue, as well as a class of scheduled processes. The file has lines that have one of two forms:

1. schedule *process deadline duration*, or
2. run *time*.

where:

- *process* is an arbitrary string,
- *deadline*, *duration*, and *time* represent time.

For example, your program might be given a file with the following contents:

schedule lunch 1300 30
schedule work 7200 2300
schedule homework 2359 300
run 2400

It doesn't really matter what the units of time are in this file, since the scheduler works the same either way. If it helps, you can think of the first two digits as hours and the last two digits as minutes, so that in this example lunch would be scheduled to complete no later than 13:00 (1pm) and 30 minutes are needed

for lunch, whereas homework would be scheduled with a deadline of 23:59 (11:59pm), and 3:00 (three hours) must be allowed to complete the homework (note that for most people, 3 hours is not enough to complete this project).

Each line beginning with schedule tells your program to insert the corresponding item into the priority queue, with a priority corresponding to its deadline. Items with earlier (smaller) deadlines should be executed first -- this is called Earliest Deadline First scheduling, and is often used in real-time systems (you can read about it in Wikipedia if you are interested).

Your program must have a long integer variable to represent time. That variable is initially set to zero, and is increased as the program runs.

Each line beginning with run *time* tells your program to advance your internal time variable until it reaches *time*. You do this by repeatedly removing the first item from the priority queue, and advancing the internal time variable by the *duration* of the process, until the *time* is reached. If the process completes before the *time* is reached, a new process is taken from the priority queue. If there are no processes left in the priority queue, the internal time is simply set to *time*. If, on the other hand, the current process would run past *time*, it is re-inserted in the priority queue with a duration equal to the remaining time.

So in the example above, time starts at zero, and when your program reads the line that says

run 2400

it starts to run the first process in the queue, which will be "lunch". Lunch takes 30 minutes, so your program will remove "lunch" from the queue and set its time variable to 30. The next item removed from the queue will be "homework" (with an earlier deadline than "work"), which takes time 300. So, the time variable is set to 330.

Finally, "work" is removed from the queue. The time is now 330, and work is scheduled to take 2300 time units. But, this run ends at time 2400, whereas if all the work was done now, the run would end at time 2630. So, the time variable is set to 2400, and your program must re-insert "work" into the priority queue with a deadline of 7200 (the deadline hasn't changed) and a duration of 230 (which is 2300 - (2400 - 330), the duration minus the difference between the end time and the start time).

Every time a process is removed from the queue, your program should print
current time: busy with *process* with deadline *deadline* and duration *duration*, where *current time* is the value of the time variable when the process is first removed from the queue.

When a process completes, your program should print
current time: done with *process*. If the process completes after its deadline, your program should print
current time: done with *process* (late).

Every time a process is inserted into the queue, your program should print
current time: adding *process* with deadline *deadline* and duration *duration*.

Some examples

Given the above example file, your program should print:

0: adding lunch with deadline 1300 and duration 30
0: adding work with deadline 7200 and duration 2300
0: adding homework with deadline 2359 and duration 300
0: busy with lunch with deadline 1300 and duration 30
30: done with lunch
30: busy with homework with deadline 2359 and duration 300
330: done with homework
330: busy with work with deadline 7200 and duration 2300
2400: adding work with deadline 7200 and duration 230

For another example, suppose this was your schedule:

schedule work 60 30
schedule fun 40 30
run 50
run 100

Your program should schedule fun before work (because in this case fun has an earlier deadline), and print the following:
0: adding work with deadline 60 and duration 30
0: adding fun with deadline 40 and duration 30
0: busy with fun with deadline 40 and duration 30
30: done with fun
30: busy with work with deadline 60 and duration 30
50: adding work with deadline 60 and duration 10
50: busy with work with deadline 60 and duration 10
60: done with work

Some suggestions

There are two kinds of time in this program. The heap methods must keep track of how long a method executes. This is actual time or elapsed time is requested from the system clock.

Your program keeps track of the time in the schedule, which is simulated time. Do not confuse the two! The elapsed time is in nanoseconds. The simulated time is in arbitrary units. Note that your program should be general i.e. that it should work for all input files in the prescribed format and not just only for the example file. At demonstration time, a different example file can be given.

Write all the algorithms (in pseudocode form) that you implemented in your program in an MS WORD document and perform the detailed complexity analysis (Find the best/average/worst cases).

Submission Instructions

1) Submit the zipped assignment folder (having the report and **all source code files only**) on the course LMS by the deadline. Name the zipped file as [your roll-number]_P1_Spr_2021.
2) E-mail the zipped file (having the report and all source code files only) to the course instructor by the deadline. You must CC the email to yourself for verification of the latest and correct submission of the project.