Computer Science - Job Scheduler

Description

Design a component called "Director" for a system called Enigma. Enigma has a pool of CPUs which execute all jobs sent to it. All the jobs are sent to Director which schedules them to run on a free CPU from the pool. A CPU can only execute one job at a time. If the number of jobs waiting to be executed exceeds the number of available CPUs, then Director decides which job should be executed first based on their priority.

A job description consists of following fields:

- Process ID : an integer in range [0, 2^63 - 1]
- Timestamp : an integer in range [0, 2^63 - 1], milliseconds since epoch
- Originating system (Varying length alphanumeric string denoting the name of the system which submitted the job)
- Encoded job instructions (Varying length alphanumeric string which each CPUs understands to decode and execute)
- Importance : an integer in range [1-100]
- Duration : an integer in range [0, 2^31 - 1], number of milliseconds it takes to execute the job

Director assigns a priority to each job based on its importance, timestamp, and duration fields in this order. If importance of two jobs are different, then the job with higher importance is given higher priority than the other. If importance is equal, then timestamp field is considered in which the job with lower timestamp is given higher priority. If timestamp is also equal, then the job with lesser duration is given higher priority. You can assume that there won't be two jobs which have all 3 fields equal.

Though Enigma is a real-time system and each CPU notifies Director when it becomes free, for the simulation purposes, we will assume that each job has fixed duration given in its description. Using this field, Director keeps track of busy and free CPUs and uses that information to route new jobs whenever it receives a assign command described below.

For historical analysis and debugging purposes, Director also supports querying its state as of a given timestamp (milliseconds since epoch).

Director reads following commands from stdin and responds by writing to stdout. It should terminate when it reaches end of file.

- cpus `<c>`
  - Specifies number of CPUs in the pool.
  - This is always the first line in the input.
- job `<timestamp> <id> <orig> <instr> <importance> <duration>`
  - This represents a job submission to the Director. Any number of jobs can be submitted.
  - Timestamps are guaranteed to be in ascending order. However, duplicate timestamps may be present.
  - Doesn't output anything for this command.
- assign `<timestamp> <k>`
  - Assigns $min(k, f, s)$ highest priority jobs as of given `<timestamp>` to free CPUs and removes them from queue, where f is the number of free CPUs at that timestamp and s is the number of

jobs waiting in the queue.

  - Timestamp is guaranteed to not be in past or future, i.e. once you have seen a assign command for a timestamp t, there won't be any job commands for timestamp less than or equal to t in future. Also, `<timestamp>` is guaranteed to be greater than or equal to the timestamp of the last job or assign command seen so far.
  - Outputs $min(k, f, s)$ lines in the order of decreasing priority of jobs, where each line is of the following format: job `<timestamp> <id> <orig> <instr> <importance> <duration>`
  - If there is no job present in the queue, then it should not output anything (not even a newline).
- query `<timestamp> <k>`
  - Similar to Assign, but for historical querying purposes.
  - Outputs $min(k, s)$ lines in the order of decreasing priority of jobs, where s is the number of jobs waiting in the queue at given `<timestamp>`, i.e. any job submission and assign commands with timestamp less than or equal to `<timestamp>` have been processed.
  - Timestamp is guaranteed to not be in future, i.e. once you have seen a query command for a timestamp t, there won't be any job or assign commands for timestamp less than or equal to t in future.
  - Each line is of the following format: job `<timestamp> <id> <orig> <instr> <importance> <duration>`
  - If there is no job present in the queue at the given timestamp, then it should not output anything (not even a newline).
- query `<timestamp> <orig>`
  - Outputs all the jobs with given originating system `<orig>` present in the queue at the given `<timestamp>`, in the order of decreasing priority of jobs.
  - Timestamp is guaranteed to not be in future, i.e. once you have seen a query command for a timestamp t, there won't be any job or assign commands for timestamp less than or equal to t in future.
  - Each line is of the following format: job `<timestamp> <id> <orig> <instr> <importance> <duration>`
  - If there is no such job, then it should not output anything (not even a newline).

Evaluation Criteria

Your submission will be evaluated on following parameters:

- Correctness and efficiency. A good solution should be optimized w.r.t. both time and memory.
- Documentation
  - You must describe the approach, data structures, and algorithms at the top of the code. It must describe any assumptions you have made and time complexity for each of the commands described above. You can also include your design choices, why you preferred data structure A over B etc.
  - Your code must have comments describing it.

Programming Languages

- You are restricted to use C, C++, Java, and Python for this problem statement.

Input Format

cpus 100

job timestamp1 id1 orig instr importance duration

job timestamp2 id2 orig instr importance duration

assign timestamp k

query timestamp k

job timestamp3 id3 orig instr importance duration

query timestamp orig

.

.

Note that job, assign, and query commands can be in any order and can occur any number of times.

Constraints

- `<c>` - an integer in range [0, 2^63 - 1]
- `<timestamp>` - an integer in range [0, 2^63 - 1]
- `<id>` - an integer in range [0, 2^63 - 1]
- `<orig>` - Varying length alphanumeric string
- `<instr>` - Varying length alphanumeric string
- `<importance>` - Unsigned integer in the range 1-100 inclusive
- `<duration>` - an integer in range [0, 2^31 - 1]
- Though orignating system and instructions are arbitrary strings, the number of distinct originating systems and distinct instructions are significantly less than number of jobs for larger test cases.
- Given Enigma is a real-time system, any look-ahead in the input file is not allowed. You must read one line of input and do the required processing before proceeding to next line of input.

Output Format

job timestamp1 id1 orig instr importance duration

.

.

Sample Input

cpus 10

job 1 1 orderProcessor BExvg 1 1

job 1 2 orderProcessor XfGie 2 2

assign 2 1

job 10 1 marketDataProcessor QfyIRX 100 100

assign 10 2

query 1 2

query 2 1

query 11 1

query 3 orderProcessor

Sample Output

job 1 2 orderProcessor XfGie 2 2

job 10 1 marketDataProcessor QfyIRX 100 100

job 1 1 orderProcessor XfGie 1 1

job 1 2 orderProcessor XfGie 2 2

job 1 1 orderProcessor BExvg 1 1

job 1 1 orderProcessor BExvg 1 1

job 1 1 orderProcessor BExvg 1 1

Explanation

For the first assign command in the input, the output generated is

job 1 2 orderProcessor XfGie 2 2

Because at timestamp 2, the number of jobs in the queue is 2 and this particular job is of higher priority than the other job, since its importance is 2.

For the next assign command, the number of jobs to be retrieved is 2, and hence the output contains both the jobs in the queue at that point of time in the order of decreasing priority.

For all the query commands, the output is as per the state of the queue at the given timestamp.

Note that there isn't any output line corresponding to "query 11 1" as there are no jobs in the queue at timestamp 11. The queue was drained fully after the assign operation at timestamp 10.