CS-350 - Fundamentals of Computing Systems
Homework Assignment #6

Due on Nov. 3, 2022 11.59 pm — Late deadline: Nov. 5, 2022 at 11.59 pm at 11:59 pm

Prof. Renato Mancuso

Renato Mancuso

Problem 1

You have been hired as a Boston MBTA platform inspector. Your job is to respond to requests for inspection of the platform conditions at specific stations.
To do your job, you are given a mobile app where you can receive requests for inspection when they are sent by the users. Once a request for platform inspection is received, you are expected to respond to it at some point in the future, but not necessarily in the order received. Your boss wants you to be smart about answering these requests so that you are able to satisfy as many as possible. As a reward for swiftly completed inspections, your boss will foot the bill for a whole pint of rosted marshmellow frappuccino at the end of your shift.
Since you are familiar with the B-line, you have been assigned to service all 24 stations on that line, which are shown at `https://mbta.com/schedules/Green-B/line?direction_id=0` (Government Center is stop #0 and BU is stop #8.) The following assumptions hold: (i) once you make it to a station, the amount of time to service the request is negligible—all you have to do is take a picture with your mobile phone, and (ii) from any station, you can travel in either direction.

a) What class of scheduling strategies are best suited to determine the order in which you should handle inspection requests? Motivate your answer.

b) To minimize the danger from switching platforms, you decide to minimize the number of times you switch directions. What scheduling strategy is consistent with that request?

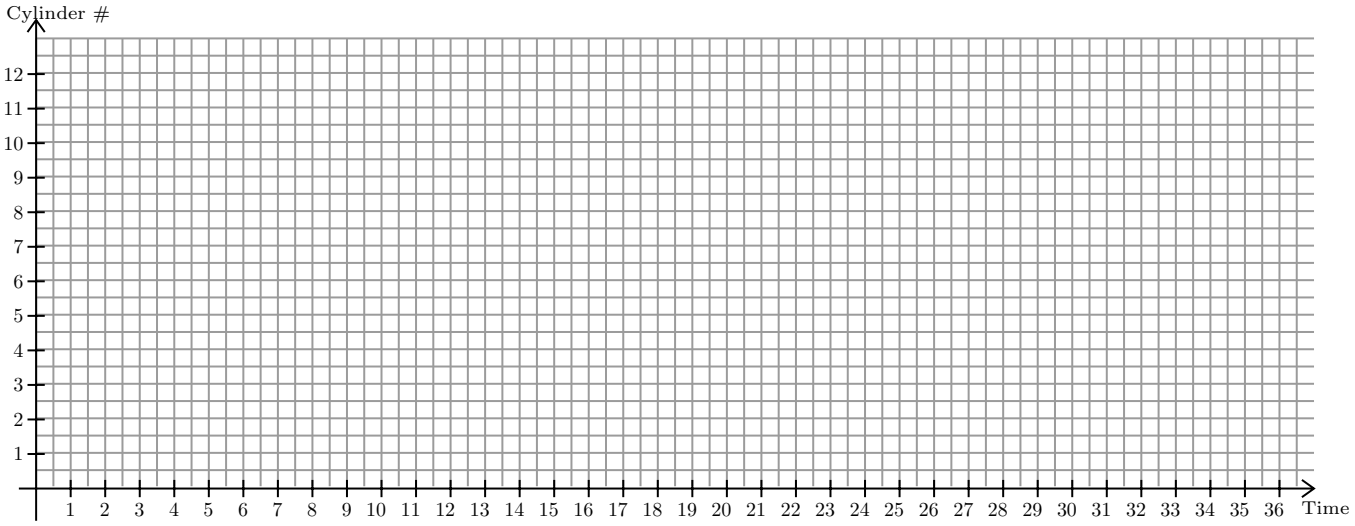
c) Now, consider the following set of platform inspection requests currently on your mobile app after inspecting platform #14 (Harvard Ave) while going eastbound (towards BU): #17, #6, #13, #19, #2, #15, #16, #0. In what order will you serve these requests?

d) You have received a complaint that you let many requests linger for too long because you were responding to these requests "out of order". Your boss now wants you to modify your strategy by not allowing more than 3 requests to go ahead of a pending request received before them. Describe your modified scheduling strategy and apply it to the scenario in Part c).

Problem 2

A disk has 12 cylinders and that are concurrently used by 4 processes. Requests coming from process P1 always target cylinder 10 and arrive every 12 time units. Requests form P2 target cylinder 7 and arrive every 6 time units. Requests from P3 target cylinder 5 and arrive every 5 time units. Requests from P4 target cylinder 2 and arrive every 18 time units.
Assume the following: (1) the first request of each task arrives at time 0; (2) the disk head is able to move by TWO cylinders every time unit; (3) when the disk head is in the right position, it takes ONE extra time unit to complete serving a request. During this time, the disk head cannot move; (4) only after a request has been fully served, a new scheduling decision can be made; (5) the disk head is initially positioned at cylinder 2.

a) Use the grid below to visualize the schedule produced by the Shortest Scan Next scheduling algorithm until time 36. *Carefully* read the 5 rules described above.



b) Use the same style of grid as in the previous Part a) to visualize the schedule produced by the SCAN scheduling algorithm until time 36. Assume that the disk head travel direction at time 0 is UP. *Carefully* read the 5 rules described above.

c) Use the same style of grid as in the previous Part a) to visualize the schedule produced by the C-SCAN scheduling algorithm until time 36. *Carefully* read the 5 rules described above.

d) Use the same style of grid as in the previous Part a) to visualize the schedule produced by a static priority scheduler that uses the Rate Monotonic rule to assign a fixed priority to each process. Draw only until time 36. *Carefully* read the 5 rules described above.

e) Which algorithm appears to exhibit the best performance if the objective is to minimize the overall distance traveled by the disk head between time 0 and 36 while serving the largest number of requests? Motivate your answer.

Problem 3

The engine controller of a Boeing 747 airliner is comprised of four control tasks, namely the Fuel Injector (FI) task, the Spin Rate Controller (SR), the Air Pressure Regulator (AP), and the Anomaly Detector (AD). The FI, SR, AP, and AD tasks are released every 7, 9, 12, and 17 milliseconds, respectively. The distribution of runtimes of the 4 tasks has been estimated. The FI task has a runtime of 1.5±0.5 milliseconds, i.e. a mean runtime of 1.5 milliseconds, a min runtime of 1.5-0.5 milliseconds, and a max runtime of 1.5+0.5 milliseconds. The SR task has a runtime of 1.8±0.2. The AP task has a runtime of 2.3±0.7 milliseconds. Finally, the AD task has a runtime of 3.9±0.1 milliseconds.

a) Consider only the system comprised of tasks FI, SR, and AP. Is the system schedulable on a single-CPU processor using RM? Motivate your answer.

b) From now on, consider the system with all the 4 tasks. Is the system schedulable on a single-CPU processor using RM? Motivate your answer.

c) Is the system schedulable on a single-CPU processor using EDF? Motivate your answer.

d) Draw the schedule produced by EDF until time 34.

e) Starting from time 0, at what time in the future the same exact pattern of task activations and executions will repeat? *Hint: there is no need to draw anything. Just think about it.*

f) Consider a 2-CPU processor which is slower than the one considered in the original system. In particular, in the new processor, each task takes 60% more time to execute — but the periods do not change! Is it possible to schedule the system by first splitting the 4 tasks across the 2 CPUs and then using RM on each CPU? Explain your reasoning.

Problem 4

Code: In this problem you will write two functions to navigate complex structures that are based on the ability to crack MD5 hashes.

a) Extend the class `Dispatcher.java` that you implemented as part of HW5 to parallelize the cracking of a set of hashes. Use Java support for multi-threading to define as many woorker threads as the number of CPUs N in the machine—passed as a parameter to your code. The Dispatcher should construct a global work queue where each hash to crack is a unit of work and provided to the first available worker for cracking. Define a `dispatch(...)` method to handle the distribution of work units to worker threads.

Apart from implementing the `dispatch(...)` method, the class should also include a `public static void main(String [] args)` function. The `main(...)` function should accept 2 parameters from the calling environment. First, the input data (list of hashes to crack) is given in a file. The path to the file is passed as the first parameter by the calling environment. The input file is structured as a list of MD5 hashes to crack, one per line, with each line terminated with a newline (\n) character. Second the number of CPUs available on the machine N is passed as the second parameter.

It is responsibility of the `main(...)` function to internally initialize and start the dispatcher. It is the job of the dispatcher to (1) read the input file; (2) distribute the work to the worker threads and let the threads print the result. The result of each of the unhash operations, i.e. the cracked hashes, should be printed in output, with a single line per decoded hash. Apart from the list of decoded hashes, nothing else should be printed in output.

b) Modify the dispatcher to be able to handle impossible hashes. For this part, you will be given an input file that contains a few hashes that are impossible to crack. Extend the dispatcher to handle timeouts in the processing of hashes so that the workers do not get stuck trying to reverse impossible hashes.

For this part, the input file will appear identical to the previous part, at least on the surface. The second parameter passed to your code will still be the number of available CPUs. A third parameter is added for this part, that encodes the (recommended) length of the timeout for impossible hashes expressed in milliseconds.

Just like the previous part, it is responsibility of the `main(...)` initialize and start the dispatcher. It is the job of the dispatcher to (1) read the input file; (2) distribute the work to the worker threads and let the threads print the result. Importantly, (3) the dispatcher should handle the timeout for all the threads. If a worker does not complete on time, the dispatcher should order the thread to move on. In this case, the uncrackable hash as it was read from the input file should be print in output.

HINT: in Java, thread signaling works in a very counter-intuitive way. Use variables to control the execution flow on the worker threads instead.

Submission Instructions: in order to submit this homework, please follow the instructions below for exercises and code.
The solutions for Problem 1-3 should be provided in PDF format, placed inside a single PDF file named `hw6`.pdf and submitted via Gradescope. Follow the instructions on the class syllabus if you have not received an invitation to join the Gradescope page for this class. You can perform a partial submission before the deadline, and a second late submission before the late submission deadline.

The solution for Problem 4 should be provided in the form of Java source code. To submit your code, place all the `.java` files inside a compressed folder named `hw6`.zip. Make sure they compile and run correctly according to the provided instructions. The first round of grading will be done by running your code. Use CodeBuddy to submit the entire `hw6`.zip archive at `https://cs-people.bu.edu/rmancuso/courses/cs350-fa22/codebuddy.php?hw=hw6`. You can submit your homework multiple times until the deadline. Only your most recently updated version will be graded. You will be given instructions on Piazza on how to interpret the feedback on the correctness of your code before the deadline.