CS-350 - Fundamentals of Computing Systems
Homework Assignment #7

Due on November 10, 2022 — Late deadline: November 12, 2022 EoD at 11:59 pm

Prof. Renato Mancuso

Renato Mancuso

Problem 1

The trajectory planning subsystem of NASA's Perseverance rover is highly critical and is comprised of 4 tasks. The first task performs motion course (MC) calculation every 10 ms, which takes at most 3.5 ms. The second is in charge of performing sensor fusion (SF) between inertial measurements and incoming video feed. SF operates at 100 Hz and its runtime varies anywhere between 3 and 7 ms. The third task is responsible for object detection (OD) and is activated at 25 Hz. OD can be carried out by analyzing the last batch of video frames, which takes 8 ms in the worst case. Finally, a diagnostic (DG) task has a period of 24 ms and a WCET of 12 ms.

a) Compute the minimum number of CPUs (if you had the perfect scheduler) necessary to consolidate all the workload necessary to perform trajectory planning.

b) Consider the tasks in the order in which they appear in this question. Is is possible to schedule the system via EDF-FF on a 2-CPU system?

c) Is it possible to use partitioned EDF to schedule the considered taskset on 2 CPUs?

d) Consider only the MG and DG tasks. Would it be possible to schedule them on the same CPU using RM?

e) Is the system schedulable under Global EDF? Motivate your answer.

Problem 2

A sensor node needs to acquire and pre-process the data from 4 different onboard sensors. It is crucial that pre-processing completes always before a new data sample is available at a given sensors to prevent loss of samples. The sensors are the Gyroscope (G), Magnetometer (M), Accelerometer (A), and Baroscope (B). The gyroscope produces data at a rate of 200 Hz, with each data sample requiring 2 ms for pre-processing in the worst case. Pre-processing for the magnetometer takes no longer than 1 ms with samples being produced at 125 Hz. A new accelerometer sample is produced every 19 ms and pre-processing takes at most 4 ms. Finally, pre-processing a barometer sample takes at most 6 ms, but the sampling period is configurable.

a) Without drawing the schedule, perform a back-of-the-envelope calculation to determine a suitable configuration for the sampling period of the barometer such that the system always operates correctly under RM. For this part, consider only values of periods that are integers when expressed in milliseconds.

b) Because the calculation above returned a value of sampling period which is too large, you decide to set the barometer sampling period to 29. Will the system operate correctly under RM?

c) You decide to set the barometer sampling period to 30 ms and stick to RM. You notice however that once in a while (very rarely) the pre-processing of the accelerometer takes 5 ms instead of its nominal 4 ms worst-case execution time. When this happens, the system can still operate correctly if **the faulty accelerator pre-processing job is killed when it misses the deadline**. Killing any other job is not acceptable. Also, you can apply a utilization-preserving transformation to all or some of your tasks. Specifically, you can release any pre-processing task twice as fast but only perform half of the required processing in each release. Explain how you can make the system operate correctly under RM.

d) Consider the same system as the question above—i.e., where the period of the barometer has been set to 30 ms. Compute the worst-case response (WCRT) time for all the jobs of each task.

e) Consider the same system as the question above—i.e., where the period of the barometer has been set to 30 ms. How much slower can the processor be while still ensuring that all the deadlines are met?

f) Consider the same system as the question above—i.e., where the period of the barometer has been set to 30 ms and the CPU has not been slowed down. Is the system schedulable under SJN?

Problem 3

Two processes, namely the Printer and the Shifter, co-operate to respectively print and update the value of an array of integers that has a fixed size of 5 elements. The code executed by the Printer process is provided in Listing 1; the code of the Shifter process is given in Listing 2. If a variable has the same name in both processes, it is to be considered a shared variable. A `[FRAGMENT x]` statement in the code refers to additional lines of code (a fragment) to be inserted in place of the statement. At the beginning of time, both processes start executing at their `start` line and will not run again once they reach their `end` line. The initial state of the shared data structures is as follows: `array = [4, 9, 5, 3, 6]`; `queue = [8, 1, 7, 2]` (NOTE: 8 is at the head of the queue.)

```
1  Process Printer:
2     start {
3        tot = 0;
4
5        [FRAGMENT P1]
6
7        for (i = 0; i < 5; ++i) {
8           tot += array[i];
9        } loop;
10
11       [FRAGMENT P2]
12
13       print(tot);
14 } end;
```

Listing 1: Code of Printer Process

```
1  Process Shifter:
2     start {
3        while (! queue.empty()) {
4           prev = queue.pop();
5
6           [FRAGMENT S1]
7
8           for (j = 0; j < 5; ++j) {
9              tmp = array[j];
10             array[j] = prev;
11             prev = tmp;
12          } loop;
13
14          [FRAGMENT S2]
15
16       } loop;
17    } end;
```

Listing 2: Code of Shifter Process

a) Assume that P2 and S2 are empty and that (a) fragment P1 expands to `wait(go);` and (b) S1 expands to `if(queue.empty()){signal(go);}`. Here, `go` is a semaphore initialized to 0. What is the maximum value that could ever be printed at line 13 by the Printer process?

b) Assume that (a) fragment P1 and S1 expand to `wait(m);` and that (b) P2 and S2 expand to `signal(m);`. Here, `m` is a semaphore initialized to 1. What is the maximum value that could ever be printed at line 13 by the Printer process?

c) Assume that fragments P1, P2, S1, and S2 are all empty. What is the maximum value that could ever be printed at line 13 by the Printer process?

Problem 4

Code: In this problem you will write a function to navigate a complex multi-level structure that has an emerging structure that is progressively revealed as the different hashes are cracked.

a) The code you have written in the previous assignments should have provided you with a basic unit, namely the Dispatcher, capable of creating a work queue and distributing the work to a set of worker threads. In this problem, we will further extend the code to introduce a super-entity responsible for coordinating the macro-phases of the processing flow. Because in the end the goal is to find a treasure, we call the coordinating super-entity the *Pirate*.

Let us first review the processing problem at hand and then define the operating scope of the Pirate entity. The file provided in input is still a list of hashes. Just like in the previous homework assignment, the list contains hashes that can be immediately cracked and result in some non-negative integer number. Other hashes are impossible to crack. And other yet are crackable with a *compound hint*. The idea is that the compound hint can be constructed from the integer values of the hashes that were immediately crackable.

Specifically, as you crack those hashes that have a simple answer, your code will produce a sequence of integers. Consider the sorted list of integers $L = \{A, B, C, D, \ldots\}$ such that $A > B > C > D \ldots$. Then, a compound hint has the form: "$\alpha; k; \beta$" (this is literally a string formed by concatenating the three numbers α, k, and β with the delimited character ";" in between!) where α and β are some integers in L and such that $\alpha < \beta$. Note that α and β are not necessarily contiguous in L. Finally, k is any integer between α and β but different from them. Moreover, no two compound hints share the same α or β. So if a compound hint was of the form "$\alpha; k; \beta$" (for some k), then there cannot exist any compound hint with the form "$\alpha; k'; \beta$" or "$\beta; k'; \gamma$" or "$\alpha; k'; \gamma$" or "$\gamma; k'; \alpha$" or "$\gamma; k'; \beta$".

Let's make an example. Suppose we have the following input file content:

```
202cb962ac59075b964b07152d234b70
d81f9c1be2e08964bf9f24b15f0e4900
03f206ba972ab43c585255d26826c4b3
37cba5957ccb647a5fb310923ac8794a
```

You will notice that the first and second hash can be reversed as "123" and "345", respectively. While the third and fourth hash appear to be initially uncrackable. We can then construct any compound hint that follows the format "$123; k; 345$" and where $k \in \{124, \ldots, 344\}$. We then discover that the fourth hash can be cracked with the compound hint "123;234;345". The output produced by the code should then be[1]:

```
123
345
123;234;345
03f206ba972ab43c585255d26826c4b3
```

Write a Java class `Pirate.java` that defines a method called `findTresure(...)` that internally uses the dispatcher to initially split the hashes in input to the worker threads. There are no restrictions on the exact arguments and return value of this method, so use whatever you deem more appropriate. The job of `findTresure(...)` is to coordinate the processing steps. After organizing the first run, it collects the result from the worker threads (cracked hashes and timed out hashes) and organizes a new run. In the second run, it will consider each pair of numbers α, β in the resulting L list and use the

[1] The order of the lines in the output is not important.

worker threads to check if any timed-out hashes can be cracked with a compound hint constructed using α and β.

Apart from implementing the `findTreasure(...)` method, the class should also include a `public static void main(String [] args)` function. The `main(...)` function should accept 3 parameters from the calling environment. The path to the file is passed as the first parameter by the calling environment. The format of this file is the same as in the previous assignments. The second parameter passed to your code is the number of available CPUs. The third parameter encodes the length of the timeout for (currently) uncrackable hashes expressed in milliseconds.

It is responsibility of the `main(...)` function to internally invoke the implemented `findTreasure(...)` method **only once** and make sure that the desired result is printed in output.

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 `hw7`.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 `hw7`.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 `hw7`.zip archive at `https://cs-people.bu.edu/rmancuso/courses/cs350-fa22/codebuddy.php?hw=hw7`. 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.