CS-350 - Fundamentals of Computing Systems
Homework Assignment #8

Due on December 3, 2022 — Late deadline: December 5, 2022 EoD at 11:59 pm

Prof. Renato Mancuso

Renato Mancuso

Task ID	Period (ms)	WCET (ms)
1	10	5
2	12	2
3	19	5

Table 1: Real-Time Taskset.

Problem 1

Consider the Taskset provided in Table 1 to be scheduled using RM on a single-processor system.

a) First, show that the taskset is schedulable under RM.

Consider now the case in which Task 1 and Task 3 share a resource of unitary size (mutually exclusive). Every time a job of **Task 1** runs, it will attempt to acquire the resource by performing a `wait(sem)` on a shared semaphore after exactly 1 ms into the job's execution. It will then hold the resource for exactly 2 ms before releasing it through a `signal(sem)`. It will then run for an additional 2 ms before completing — hence the total WCET is 5 ms, as reported in Table 1. For additional clarity, the code of Task 1 is reported in Listing 1

Listing 1: Code of Task 1.

```
Process Task_1:
    /* Global Variables */
    semaphore sem = 1;

    every 10 ms:
       /* REMINDER SECTION 1 - LENGTH = 1 ms */

       wait(sem);

       /* CRITICAL SECTION - LENGTH = 2 ms */

       signal(sem);

       /* REMINDER SECTION 2 - LENGTH = 2 ms */
    done
```

Every time a job of **Task 3** runs, it will attempt to acquire the resource by performing a `wait(sem)` on a shared semaphore after exactly 0.5 ms (length of reminder section 1) into the job's execution. It will then hold the resource for exactly 4.5 ms (length of critical section) before releasing it through a `signal(sem)`, and complete right away — hence the total WCET is 5 ms, as reported in Table 1.

b) Show that the system is not schedulable anymore under RM. *Hint: produce the schedule until at least time 31 ms.*

c) Would the system be schedulable under EDF? *Careful: the utilization test studied in class applies only to tasks that do not share resources!*

d) Read this article and then answer the following: how would you fix the issue that leads to the system not being schedulable under RM? Show the schedule produced by your solution until at least time 38.

Problem 2

In a small island in southern Italy, the city of Semaforia exists. In Semaforia, democracy works in a peculiar way. In fact, over there, there is a Major and a City Council. The Major listens to the complaints of the citizens and proposes ordinances to be voted by the council. The council then comes together and each person on the council needs to vote with a YES/NO. For the voting to start, it is fundamental that ALL the councillors are inside the council room. Once they are all inside the council, each of them can cast their vote. After all the votes have been cast by all the councillors, the votes are counted by the Major. If there are 50%+1 votes in favor, the ordinance is passed, otherwise it is rejected. The council cannot vote on a different ordinance until the current one has been fully processed. The council has N councillors.

Listing 2: Template code for Major and Councillors.

```
Process Major:
   Repeat:
        listen_to_citizens()
        prepare_ordinance()

        /* Add here the code to trigger a new action of the council, */
        /* and to gather all the votes.                              */

        if (yes_votes > N*0.5 + 1) {
           make_ordinance_official()
        } else {
           destroy_ordinance()
        }
   Forever


/* The following is instantiated N times. */
Process Councillor:
   Repeat:
        /* Block until no new ordinance to vote on is available */

        read_ordinance()

        get_to_the_council()

        /* The following should happen only when all the councillors */
        /* are in the council.                                       */

        if(liked_ordinance()) {
            /* Vote YES */
        } else {
            /* Vote NO */
        }

        /* When all the votes have been cast, only ONE of the   */
        /* councillors is responsible for the following action. */

        notify_major()

   Forever
```

a) Your first job is to implement the logic of Semaforia using semaphores by defining two types of processes. The first type of process represents the Major and is only instantiated once. The second type captures the behavior of the generic Councillor. Start from the templates provided in Listing 2. Introduce any variable and semaphore that you may deem necessary, specifying how each is initialized. Your code should never perform busy-waiting.

b) It turned out that democracy in semaforia worked very slowly. The problem is that voting cannot start before all the councillors are in the room. But while some councillors showed up almost immediately, others arrived at the council a few days after everyone else. As such, the voting scheme is modified.

Problem 2 continued on next page. . .

Out the total N councillors, only some $K < N$ are required to be in the room before the voting starts. Actually, once K councillors are in the room, the council door is closed and nobody is allowed to enter, let alone vote on the current ordinance. Modify your code from Part (a) to implement the new logic. Be extra careful: councillors who arrived after the first K are not allowed to vote on the current ordinance; they can vote on the next one but they should always read it first.

Problem 3

Consider as system with 5 processes $P_1, \dots P_5$ sharing 3 resources $R_1, \dots, R_3$. The immutable system parameters are provided in Table 2.

		Parameter	Resources		
			R_1	R_2	R_3
		$R(k)$	10	5	7
Processes	P_1	$C_1(k)$	7	5	3
	P_2	$C_2(k)$	3	2	2
	P_3	$C_3(k)$	9	1	2
	P_4	$C_4(k)$	2	2	2
	P_5	$C_5(k)$	4	3	3

Table 2: Static parameters for the considered system.

a) Consider the state of the system at a given point in time as provided in Table 3. Complete the table with the missing parameters.

		Parameter	Resources			Parameter	Resources		
			R_1	R_2	R_3		R_1	R_2	R_3
		$V(k)$							
Processes	P_1	$A_1(k)$	0	0	0	$N_1(k)$			
	P_2	$A_2(k)$	2	0	0	$N_2(k)$			
	P_3	$A_3(k)$	3	1	2	$N_3(k)$			
	P_4	$A_4(k)$	2	1	1	$N_4(k)$			
	P_5	$A_5(k)$	0	0	2	$N_5(k)$			

Table 3: System state for considered system.

b) Determine if the current state would be deemed safe by the Banker algorithm for deadlock avoidance. Provide your reasoning for full marks.

c) While the system is in the state described by Table 2 and 3, P_1 submits an allocation request for 2 units of R_1, 1 unit of R_2, and 1 unit of R_3. Can the request be safely granted? *You may use Table 4 to answer the question.*

		Parameter	Resources			Parameter	Resources		
			R_1	R_2	R_3		R_1	R_2	R_3
		$V(k)$							
Processes	P_1	$A_1(k)$				$N_1(k)$			
	P_2	$A_2(k)$				$N_2(k)$			
	P_3	$A_3(k)$				$N_3(k)$			
	P_4	$A_4(k)$				$N_4(k)$			
	P_5	$A_5(k)$				$N_5(k)$			

Table 4: System State.

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) Extend your `Pirate.java` class to navigate a multi-level treasure map, build up the list of hints and then use the full list of hints to discover the instructions to locate the treasure!

As usual the starting point is a list of hashes. Some of these hashes can be cracked right away (simple hints), while others can be cracked with compound hints. The main difference compared to HW7 is that before we only had two processing stages: (1) initial stage for simple hints; (2) second stage for compound hints. Now, we might have multiple subsequent processing stages that build on the result of the previous stages. Let us review this concept.

Starting from the first stage, we first attempt to crack the input hashes as if they are simple hints and use the same timeout mechanism to detect so-far uncrackable hashes (just like in HW7). This leads to a first list of integers $L_0 = \{A, B, C, D, \ldots\}$. Next some of the hashes that could be not previously cracked might be crackable with a compound hint of the form "$\alpha; k; \beta$" generated in the same way as we previously did—and with the same constraints between α, k, and β.

Completing the processing described above leads to the definition of a new list of integers $L_1 = \{k_1, k_2, k_3, \ldots\}$ where each k_i comes from a different compound hint. Note that none of the integers in L_0 can show up in L_1. Using the new list of integers some more (or potentially all) of the so-far uncrackable hashes can be cracked with a new compound hint built in the same way as before. That is, using a compound hint of the form "$k_x; h; k_y$" where $k_x < h < k_y$ and $k_x, k_y \in L_1$. If after this step there are still uncrackable hashes, one more round of processing is needed by defining a new list of integers L_2 etc. In this problem, all the input hashes are crackable! Thus, a valid solution should not leave any hash uncracked.

Consider now the various list of integers $L_0, L_1, L_2, \ldots$ constructed in the multiple processing stages. To find the instructions for the treasure, we need to take the union of all the integers in the list, and then sort the integers in ascending order. Finally, each number is the offset of a single character inside a cipher-text file passed in input.

Your job is to print, in order, the corresponding characters in the second file at the correct offsets. The resulting string will provide the final directions to find the treasure. Let us make a full example. Let's say that the input list of hashes is:

```
c81e728d9d4c2f636f067f89cc14862c
c9f0f895fb98ab9159f51fd0297e236d
d3d9446802a44259755d38e6d163e820
c51ce410c124a10e0db5e4b97fc2af39
cc397dd8486d3b0aec12fd25c76b19e2
3e66ce357af2da396c17631e706f1941
dd6cf119df0327fd2fc7500985284f59
```

After the first stage we have the intermediate result below, so $L_0 = \{2, 8, 10, 13\}$:

```
2
8
10
13
cc397dd8486d3b0aec12fd25c76b19e2
3e66ce357af2da396c17631e706f1941
dd6cf119df0327fd2fc7500985284f59
```

Problem 4 continued on next page...

Next with the integers in L_0 we are able to crack two more hashes, so the partial result is the one below and $L_1 = \{5, 11\}$:

```
2;5;10
8;11;13
dd6cf119df0327fd2fc7500985284f59
```

Finally, the remaining hash can be cracked with the compound hash below:

```
5;7;11
```

Putting all the integers from the lists together, we have the following list of numbers: $\{2, 5, 7, 8, 10, 11, 13\}$. Now if the content of the input cipher-text is:

```
ljDshoDne{ :S)43 298r76QtO
```

The solution is the string: "`Done :)`", comprised by the characters in the position indicated by the integers in the final list.

Extend the Java class `Pirate.java` that defines a method called `findTresure(...)` that is able to find the location of the treasure as described above. The **only** output of the code should be the final instructions to find the treasure. Apart from implementing the `findTreasure(...)` method, the class should also include a `public static void main(String [] args)` function. The `main(...)` function should accept 4 parameters from the calling environment. (1) The path to the input hash list is passed as the first parameter. (2) The second parameter passed to your code is the number of available CPUs. (3) The third parameter encodes the length of the timeout for (currently) uncrackable hashes expressed in milliseconds. (4) The fourth parameter is the path to the cipher-text to decode to find the final instructions for the treasure.

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 `hw8`.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 `hw8`.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 `hw8`.zip archive at `https://cs-people.bu.edu/rmancuso/courses/cs350-fa22/codebuddy.php?hw=hw8`. 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.