HY486: Principles of Distributed Systems
Professor: Panagiota Fatourou

1st Set of Exercises: Theory Part

***Deadline**: Monday, 2 Nov 2020*

Graduate students should answer all the exercises in order to get the grade of 100. Undergraduate students do not have to answer the parts that are labelled as [Graduate] although they may want to do so for those exercises that are referenced as bonus for them.

Exercise A [35%]

1. Apart from ensuring mutual exclusion for the access of a set of processes to a shared resource, one can also allow the processes to access the resource in groups. For this notion of group synchronization, an algorithm considers ***k*** groups of processes. At any given moment in the execution of the algorithm, a group is considered to either be *enabled* or *disabled*. Each process that wishes to access a shared resource must first choose or be assigned to one of the ***k*** groups and then, it can proceed to use the resource once it detects that its group has become enabled.

    a. Examine the pseudocode presented in Figure 1, for a system of *n* processes. Explain how the algorithm works. Does it correctly ensure group synchronization for ***k = 2***? Explain your reasoning. [5%]

    b. Is it possible to have just the threads of one of the groups accessing the shared resource repeatedly forever, without ever allowing to the threads of the other group to access the resource (i,e., to have some sort of group starvation)? Justify your answer. [3%]

    c. What would happen if we removed lines 1 and 5 from the algorithm? [4%]

    d. What would happen if we removed line 4 from the algorithm? [4%]

Code for process p_i:

```
Shared variables:
group: atomic bit;
state[1..n]: array of atomic registers, values range over {0,1,2,3};
Initially, for each i, 1 ≤ i ≤ n: state[i] = 3;

1.    state[i] = 2;
2.    state[i] = group;
3.    for j = 1 to n {
4.          if (state[i] ≠ group) break;
5.          while (state[j] = 2) noop;
6.          if (state[j] == 1 − state[i])
7.             while ((state[j] == 1 − state[i]) AND ((state[i] == group) noop;
9.    }
9.    < access resource >
10.   group = 1 − state[i];
11.   state[i] = 3;
```

Figure 1

2. A private clinic accepts patients for Covid-19 tests. Each patient arrives to the clinic alone and acquires a unique ticket, with a constantly increasing integer number that uniquely identifies patients in the order of their arrival. [19%]

There is also a number of workers. Each worker requests from the next (in the order defined by the tickets) patient to follow him/her inside the clinic. The workers should use a counter to indicate those patients that are allowed to enter into the clinic. When a patient enters the clinic, s/he has to fill in some documents with his personal information and contact details. To do so, each patient should use one of the ***d*** available desks. Only one patient can make use of a desk. S/he randomly chooses one of them and possibly contends with other patients to get access to this desk. Use a queue lock for each desk to ensure that patients contending for it are served in a FIFO order.

After filling in all the documents, the patient moves to the next room where he has to leave all his belongings. There are ***c*** cabinets for doing so, which you need to implement using an array of Booleans. (Assume that c is big enough for all patients.) Each patient should find the first element in the array that is free and acquire it, by setting its value. The cabinet will be released before the patient leaves the clinic. Access to the cabinet array should be protected using a lock.

[**Undergraduate students**] The patient then moves to the last room where a nurse will take the samples required for the medical examination. As long as the medical examination is taking place, the cabinet is occupied and the nurse does not examine any other patient. After the completion of the examination, the patient takes his belongings and then leaves the clinic.

Provide a lock-based solution to the synchronization problem presented above.

[Graduate students] [Bonus for undergraduate students: 5%] The patient then moves to the last room where a nurse will take the samples required for the medical examination. There are enough nurses for all patients, so multiple patients can be examined at the same time. The clinic's doctor visits each patient and announces the results to the patients. The patient leaves the clinic only after the doctor visits him/her. A doctor visits the patients only after a specific number (***k***) of samples has been received by the nurses.

Assume that there are ***w*** workers in the system, which you need to implement, e.g. using an array of w positions which will describe the worker's availability. The patient with ticket i can enter the clinic and move between its rooms only if s/he is guided by the worker with id (i MOD w).

After the patient has been visited by the doctor, s/he is guided to cabinets' room and then directly to the entrance. So, in this case the patient does not cross the desks' room.

Provide a lock-based solution to the synchronization problem presented above.

Exercise B [Undergrads: 30%, Grads: 15%]

1. Is it possible to simplify the lazy list's validation method **`validate()`** by dropping the check that **`pred.next`** is equal to **`curr`**? If yes, explain your reasoning. If not, provide a counterexample. [10%, 5%]
2. Would the lazy list algorithm still work if we marked a node as removed simply by setting its **`next`** field to **`null`**? If yes, explain your reasoning. If not, provide a counterexample. [10%, 5%]
3. Can you modify the **`insert()`** method of the fine-grained synchronization list so that it locks only one node? If yes, explain your reasoning. If not, provide a counterexample. [10%, 5%]

Exercise C [Undergrads: 35%, Grads: 50%]

In this exercise, you have to design and analyze a **lock-based** implementation of a concurrent **threaded** binary search tree. In your implementation, each operation (BSTInsert, BSTDelete, and BSTSearch) should acquire a minimum number of locks on nodes during its execution. Moreover, operations occurring in parts of the tree that are independent should occur concurrently (so, the use of a single global lock is not allowed). Your implementation should use fine-grain locking in order to increase parallelism.

1. Present pseudocode for BSTInsert, BSTDelete, and BSTSearch. [20%, 13%]
2. Provide an informal intuitive description of your implementation. [5%, 3%]
3. Fix any execution α of your implementation. Explain how linearization points should be assigned to the operations in α. [5%, 3%]
4. [**Graduate students, Bonus for undergraduate students**] Argue that your algorithm is correct. [5%, 3%]
5. Argue about the progress properties of your implementation (e.g. is the algorithm deadlock-free? why? is the algorithm starvation-free? why?) [5%, 3%]
6. [**Graduate students, Bonus for undergraduate students**] Analyze the complexity of your algorithm on a cache-coherent NUMA architecture and a cache-less NUMA machine. To do so, choose a specific lock implementation, and provide bounds on the number of cache misses that occur on a cache-coherent SMP architecture, as well as on the number of remote memory references that occur on a cache-less NUMA architecture. [5%, 5%]
7. [**Graduate students, Bonus for undergraduate students**] Solve the exercise above assuming that the synchronization is lazy. [20%]

Complementary Material for threaded trees:

https://www.csd.uoc.gr/~hy240/current/material/supplementalMaterial/threadedTrees.pdf