**NAME: PURUSHOTTAM KUMAR**

**REG: 11807502**

**B-TECH CSE (2ND YR)**

**OPERATING SYSTEM ASSIGNMENT**

**GitHub link: https://github.com/puru531/OS-Assignment**

**Q)21.** A number of cats and mice inhabit a house. The cats and mice have worked out a deal where the mice can steal pieces of the cats’ food, so long as the cats never see the mice actually doing so. If the cats see the mice, then the cats must eat the mice (or else lose face with all of their cat friends). There are NumBowls cat food dishes, NumCats cats, and NumMice mice. Your job is to synchronize the cats and mice so that the following requirements are satisfied:No mouse should ever get eaten. You should assume that if a cat is eating at a food dish, any mouse attempting to eat from that dish or any other food dish will be seen and eaten. When cats aren’t eating, they will not see mice eating. In other words, this requirement states that if a cat is eating from any bowl,then no mouse should be eating from any bowl. Only one mouse or one cat may eat from a given dish at any one time. Neither cats nor mice should starve. A cat or mouse that wants to eat should eventually be able to eat. For example, a synchronization solution that permanently prevents all mice from eating would be unacceptable. When we actually test your solution, each simulated cat and mouse will only eat a finite number of times; however, even if the simulation were allowed to run forever, neither cats nor mice should starve.

**Solution ->**

**Solve a Synchronization Problem**

For this part of the assignment, you are expected to implement a solution to a synchronization problem

called the “cats and mice” problem. The synchronization primitives that you may use in your solution are

semaphores, locks, and condition variables. You are free to use whichever of these synchronization

primitives you choose, however you like. However, you must not directly use any “lower-level”

methods of synchronization, such as wait channels or spinlocks.

**The Cats and Mice Problem**

A number of cats and mice inhabit a house. The cats and mice have worked out a deal where the micecan steal pieces of the cats’ food, so long as the cats never see the mice actually doing so. If the cats see the mice, then the cats must eat the mice (or else lose face with all of their cat friends). There are NumBowls catfood dishes, NumCats cats, and NumMice mice.

**Your job is to synchronize the cats and mice so that the following requirements are satisfied:**

**1: Cats and mice should never be eating at the same time.**

In other words, if a cat is eating from any bowl, then no mouse should be eating from any bowl. Similarly, if a mouse is eating from any bowl, then no cat should be eating from any bowl. This will ensure that the mice don’t get eaten by the cats.

**2: Only one mouse or one cat may eat from a given bowl at any one time.**

**3: Neither cats nor mice should starve.**

A cat or mouse that wants to eat should eventually be able to eat. For example, a synchronization solution that permanently prevents all mice from eating would be unacceptable. So would a solution in which cats were always given priority over mice. When we actually test your solution, each simulated cat and mouse will only eat a finite number of times; however, even if the simulation were allowed to run forever, neither cats nor mice should starve. In addition to the above requirements, which are required for correctness, your solution should also be efficient, which we define as follows

**4: Your synchronization mechanism should not impose an upper bound on the number of bowls that can be in use simultaneously.**

In other words, if there are enough creatures, it should be possible for them to use all of the available bowls simultaneously

**Implementing Solution**

Our solution to the “cats and mice” problem should be implemented entirely in this file. file contains six functions, which are invoked by the cat and mouse simulation programs:

• cat before eating: called each time a cat eats, before it eats

• mouse before eating: called each time a mouse eats, before it eats

• cat after eating: called each time a cat eats, after it eats

• mouse after eating: called each time a mouse eats, after it eats

• catmouse sync init: called only once, before the cats and mice are created

• catmouse sync cleanup: called only once, after all cats and mice have finished Implement your solution to the “cats and mice” problem by re-implementing these functions. In particular, you should use the cat before eating and mouse before eating functions to make creatures wait before eating, when waiting is necessary to satisfy the synchronization requirements. For example, if a mouse is eating and a cat attempts to eat, the cat must be prevented from eating (at least) until the mouse has finished, otherwise requirement R1 will be violated. To enforce this, your implementation of cat before eating should cause that cat (thread) to block until it is OK for it to eat. In other words, cat before eating should not return until it is OK for the cat to eat, since once it returns the cat will eat. It is up to you to decide when it is OK for a cat (or mouse) to eat - this is the question that you must answer when designing a synchronization mechanism that satisfies the requirements. You may use catmouse sync init to create or initialize any synchronization primitives or variables that your solution requires. As provided to you,implements a very simple default synchronization mechanism. This default mechanism satisfies all of the synchronization correctness requirements using a single semaphore, which is used as a lock. This default mechanism is correct, i.e, it satisfies R1, R2 and R3. However, it does not satisfy the efficiency requirement (R4) because only one creature can be eating at any time, regardless of how many bowls there are. You should replace this default mechanism with an implementation of a new mechanism that is both correct and efficient.