Design

Question 1. List all of the synchronization primitives and shared variables that you have used to synchronize the cats and mice and identify the purpose of each one.

Semaphores:

- MaxBowlCount: The semaphore is initialized with the number of BOWLS input. The same semaphore is used for the cat and mouse simulation to prevent the total number of cats or mice ready to eat to not exceed BOWLS.

Locks:

- FreeBlockBowls: Allows unsed(currently not being used by a cat or a mouse) bowls number to be added safely to a queue. When a cat or a mouse is ready to eat, the first bowl number in the queue is retrieved. When it finishes, the bowl number is put back to the end of queue.
- WaitConditions: Allows the turn of each cat and mouse to be safetly determined. Principally, it is used along with the two condition variables described below to enforce mutual exclusion for the synchronization of turns of each cat and mouse.

Condition Variables:

- NoCatsEating: indicates that there are cats still eating
- NoMiceEating: indicates that there are mice still eating

Question 2. Briefly explain how the listed variables and synchronization primitives are used to synchronize the cats and mice.

WaitConditions, NoCatsEating, NoMiceEating

1) Determining which thread is going to eat:

- When a cat is ready to eat,

– we increase the count of waiting cats. (WaitCondition lock is used to protect these variables)

– If there are mice eating at that time, the cat waits on the condition variable NoMiceEating

– If there are cats eating and mice waiting, the cat waits on the cv NoMiceEating until the mice waiting have had the chance to eat first

- Similar for a mouse

MaxBowlCount

- Before a ready cats or mice eats, we check if there is at least one non-empty bowls. If there is are empty bowls, then the animal is allowed to eat.

FreeBlockBowls

- In order to keep track of which bowl is free or not, we have a queue of free bowls, and we assign the animal which is a going a bowl number by retrieving the first bowl number from the queue.

Question 3. Briefly explain why it is not possible for two creatures to eat from the same bowl at the same time under your synchronization technique.

We used a semaphore count `MaxBowlCount` to keep track of the number of bowls which are free or not. While the bowls are all occupied, no one is allowed to eat even if it is its turn.

To prevent two creatures from eating from the same bowl, we keep a queue of all available free bowls. Bowls are added to the queue when a creature finishes eating, and bowls are retrieved when a creature is going to eat. Locks "`FreeBlockBowls`" are used to prevent another piece of code from modifying the queue when one is already updating it.

Question 4. Briefly explain why it is not possible for mice to be eaten by cats under your synchronization technique.

When a mice is ready to eat, the code first checks that there are no cats eating at that time. If there are, then the mice will have to wait until all the cats finish eating. This is done by checking if the cats-eating-count is zero. While it is not zero, this means that cats are still eating, and so the mice are made to wait on the condition variable NoCatsEating.

Question 5. Briefly explain why it is not possible for cats or mice to starve under your synchronization technique.

Cats and mice take turns to eat. If there are cats waiting to eat and cats currently eating, then the next batch of creatures allowed to eat are the mice. If there are no mice waiting then those cats are allowed to eat. This way ensures that no creature type monopolizes all the bowls each time.

Question 6. Briefly discuss the fairness and efficiency of your synchronization technique, and describe any unfairness (bias against cats or mice) that you can identify in your design. Did you have to balance fairness and efficiency in some aspect of your design? If so, explain how you chose to do so.

Efficiency: The solution implements efficiency in this way - When it is cat turns for instance, and there are n empty bowls and $\geq n$ cats ready to eat, then n cats are sent to eat from the n bowls. The remaining cats have to wait on a semaphore count until one of the cats finish eating. So as many cats as possible are sent to eat, as long as there are available bowls.

Fairness: We ensure that every creature is treated fairly by assigning turns.