Adrian Carolli
#20381731

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.

Bowls

- char *bowlStates - array to determine which bowls are occupied contains values -, c, and m
- int availableBowls - the number of available bowls not being eaten by cats nor mice
- lock lockBowl - used to protect the bowlStates and availableBowls
- cv cvBowl - used to wait on any availableBowls

Cats and Mice

- int CAT_TIME, int MOUSE_TIME - the time it takes for a cat / mouse to eat and sleep used to optimize efficiency and fairness
- bool done - used to determine if cats or mice have all eaten
- int ALLOWED_TIME - a computed value that determines the amount of time allowed for each “round”[1] i.e. after how much time we should switch between cats and mice or vice versa
- bool catsEat - used to determine who is eating, if true cats are otherwise mice are
- int timeTaken - the time taken for each round so far, used for switching between cats and mice
- int CATS_EATING, int MICE_EATING - the number of cats / mice currently eating, used to protect mice / cats from eating
- lock lockRound - used to protect round variables i.e. done, catsEat, timeTaken, CATS_EATING, MICE_EATING
- cv cvRound - used to wait on cat or mice eating so we can start the next round

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

Cats and mice are synchronization by protecting cats from mice and vice versa by using a CV that waits on catsEat (are cats eating?) and the number of mice or cats still eating in the previous round. If this conditional is true, we allow the cats or mice to eat. We switch between rounds by changing catsEat = [true|false] when its the fair and efficient time to switch.

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.

[1] We define a “round” to be the interval that cats eat or the interval that mice eat. A new round begins when a new specie is eating and ends when that specie is no longer allowed to eat anymore.

It is not possible for two creatures to eat from the same bowl because of the bowlStates variable and its use with a CV and lock. We store the states of the bowls and when a species wants to eat from a bowl it will check for the next available bowl by searching the bowlStates array for an empty bowl. We also use a CV and lock to protect the number of available bowls as we want to only allow a specie to eat from a bowl when there is an available bowl.

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

Mice cannot be eaten by cats because of the CV that protects cats from mice. The CV waits on the number of cats eating and catsEat (whose turn is it).

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

Before the program runs it determines the ALLOWED_TIME for each round of cats or mice (the equation is shown below). After the ALLOWED_TIME is exceeded the round ends and we switch from cats to mice or vice versa. This gives cats and mice an allocated time to eat and therefore not starving them.

*ALLOWED_TIME = CAT_TIME*NumBowls;*
or
*ALLOWED_TIME = MOUSE_TIME*NumBowls;*

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.

The fairness and efficiency is determined by finding the optimal time for each round. The time is found by computing the ALLOWED_TIME based on the time it takes a cat or mouse to eat. Suppose cats take a longer time to eat and sleep, then we set the ALLOWED_TIME to be the CAT_TIME*NumBowls and we allow as many cats and mice to eat within that interval for each round.

Although, this is a simple solution it has some unaccounted for bias which can be solved. Suppose cats take 9 seconds to eat and sleep; and mice only take 8. Mice are favored because the ALLOWED_TIME is 9 seconds. So, when the mice eat they will finish eating in 8 seconds, but since they haven't exceeded 9 seconds of eating time, a new set of mice will start eating from all the bowls, allowing them an extra 8 seconds of eating time per round. This can be balanced by rounding the time it takes a specie to the ALLOWED_TIME. So, if a specie takes 8 seconds to eat since that number is so close to the ALLOWED_TIME, 9 seconds, it will stop eating and lose only 1 second per round which is much more fair. This is how we balance

fairness.

The efficiency aspect is created by allowing NumBowls of cats and mice to eat for each round at all times. There will always be NumBowls species occupying the bowls because we allow the species to all start and end at the same time.