1) ***BowlLock* (Lock)** – Used to synchronize access to all bowls. Prevents multiple threads from trying to get a bowl at the same time.

   ***BowlCv* (Condition Variable)** – Used to put threads to sleep while it isn't their "turn" to eat (ie, its not a mouse's turn while there are still cats eating), or all the bowls are in use.

   ***BowlQueue* (Queue)** – Queue of all the bowl numbers that are free for use.

   ***CurrentTurn* (int)** – Indicates which animal is allowed to use bowls (can be equal to any of TYPE_NULL: no turn is specified, TYPE_CAT and TYPE_MOUSE)

   ***isSwapping* (int)** – If it is 0 then we are not swapping turns. Otherwise it prevents new animals from going to eat at the bowls.

   ***NumServed* (int)** – Keeps track of how many consequtive animals have eaten since there have been animals of the opposite type waiting.

   ***NumEating* (int)** – Keeps track of how many animals are currently at bowls.

   ***NumWaitingForBowl* (int)** – Number of animals waiting for a bowl to free up.

   ***NumWaitingForTurn* (int)** – Number of animals waiting for the turn to switch.

2) *BowlLock* and *BowlCv* are responsible for synchronizing the process of an animal going to a bowl to eat.
   BowlQueue ensures that animals will not go to the same bowl to eat.
   CurrentTurn, isSwapping and *numEating* are responsible for keeping track of which type of animals are allowed to go to a bowl. If isSwapping is true, no new animals can go to get food until it has switched (which corresponds to the other type of animal getting a turn).
   NumServed, numWaitingForBowl and *numWaitingForTurn* are all responsible for determining when turns need to be changed. For instance, if there are more animals waiting for a turn than there are waiting for a bowl we will switch turns. If too many animals are served in a row it will also switch turns.

3) *BowlQueue* is initialized with all the bowl numbers. Whenever an animal wants to eat from a bowl it has to wait until there is at least one item in the queue. *BowlLock* and *BowlCv* make sure that only one thread is going through this section of the code at a time. When an animal finishes eating, its bowl is returned to the queue and can be used by another animal.

4) *CurrentTurn* ensures that only animals of one type can claim bowls at a time. This means that for a period of time only cats can get bowls and vice versa. However, this switches back and forth. When *isSwapping* is equal to 1, it means that the turns have been switched but there may still be animals of the other type at the bowls. When *numEating* reaches 0, it means that all the bowls are free indicating that it is safe for the animals to come and eat. In this manner cats and mice can never be at the bowls at the same time and thus, cannot be eaten.

5) It is not possible for cats or mice to starve because of the method of choosing when to switch turns. If more of one animal are waiting for their turn than there are waiting for a dish, the turns are switched around. If one type of animal has had too many consecutive feedings while the other type is waiting, it changes turns.

6) If there are significantly more cats than mice or the other way around, my technique will bias slightly more towards the one with more. This is because one of the reasons for allowing one type of animal to have the bowls is if there are more of that type waiting. If there is 1 mouse, 40 cats and 3 bowls there are very few cases where there will be more mice waiting in line than cats. However, if one or more mice have been waiting while 2*NumBowls cats have eaten it will automatically switch turns, but this will still probably only let 1 mouse eat due to the previously mentioned condition.

   Overall I would consider my technique to be quite efficient. It is constantly trying to flush out the larger queue of animals in order to try to fill up as many bowls as possible.

   My initial idea is more or less the same as the final product. I don't know of any straightforward ways of increasing both efficiency and fairness from what I have now. I believe that it is a good mix of both.