1: Synchronization primitives:
lock(lock) and condition variable(bowls_free).
Shared variable:
eating_mice,eating_cat,sleep_cat,sleep_mice,priority,bowl_usage

2:Lock ensures that only one thread can be in critical section at a time. It prevents context switches. For example when a cat thread trying to eat, it will first check whether

1: enough bowls are ready to use(check equality of eating_cat and NumBowls)?

2: mouse is currently eating(check value of eating_mice)?

3: there's one situation that if a cat is waken up.In such situation,we first check is it the cats' turn to eat?(Priority does this job), then check if there are other mice also get waken up(check value of sleep_mice). If yes, cat continues sleeping, give the chance to mice to eat. In fact, mice and cats will eat alternatively.

If either answer is yes, then put this thread on sleep(by calling cv_wait). Else, try to find the first available bowl (by checking bowl_usage), then let cat eat. Finally, all the cats finish eating(eating_cat is equal to 0), when the last cat leaves, calling cv_broadcast to notify others to eat.

This is the same when a mouse is trying to eat.

3: when a creature is trying to eat, by checking bowl_usage, it can guarantee that no two creatures can eat the same bowl.

4: when mice are trying to eat, it will first check whether cats are currently eating (checking eating_cat), if yes, let mice sleep . It ensures that cat and mouse will not eat at the same time.

5: Priority ensures that no starvation will occur. For example, if some cats finish eating in the first round, all mice waiting for the bowl(sleep_mice) will get the highest priority; all waiting cats are forced to sleep in the next round.

6:
Basically i think my implementation balances the fairness and efficiency.

For fairness, mice and cats are guaranteed to eat alternatively(unless one animal does not need to eat anymore)

So if the number of cats and mice are smaller than the number of bowls, both cats and mice are happy.

However, there is such a situation that if the number of mice is greater than the number of bowls, the remaining mice will still need to continue waiting for the next round after cats finish eat under my implementation. It is biased because it increases the waiting time for some mice. But if i let the remaining mice wait, the waiting time for cats will also increase. Because i don't know the eating time for cat and mouse; to balance fairness, i decide to let mice and cats eat alternatively. (the situation is the same if the number of cats are greater than the number of bowls)

For efficiency, I can ensure that all the bowls will be made the best use. Specifically, if it is mice's turn to eat, i will let all the mice who are in the waiting queue to eat. The situation is the same for cats.

there is also a potential efficiency issue: cats and mice need sleep. Specifically, for example, if there are 3 cats in waiting queue, 2 cats sleeping, and 5 bowls. At this time, i can just let the waiting cats eat. So here is the problem: should i wait the sleeping cat become active then let all the cats eat together? Maybe it can be more efficient but maybe it cannot. Unfortunately i have no idea about the sleeping period. So i deicide to let waiting cat eat first, then let the sleeping cat eat second. (must obey the fairness rule described above)