Caleb Williams

CSCI 455 Project 1

Which criterion of the critical section problem is not satisfied by the initial implementation? Explain.

* Because playroom-1.c is working with so few animals, it is much easier to prevent issues involving the critical sections. My implementation involved signaling that there are either ‘no dogs’ or ‘no cats’ when the final dog or cat leaves the room and keeping a pthread\_cond\_wait inside of a while loop. With this implementation and only two animals, the implementation is mutually exclusive, progress, and the bounded waiting requirements.

Describe the cats-and-dogs synchronization problem in terms of the events that each pet must wait for.

* It is relatively simple because the dogs only need to wait until there are no cats and vice versa. This makes sending the signal that the last cat or dog has left effective. The dog or cat skips waiting if the room is empty.

Do you need to add another synchronization construct to the implementation to handle birds? Explain.

* You don’t need to add another synchronization construct because you can treat birds as dogs because the birds and dogs are compatible, and the cat is compatible with neither.

Why would using a broadcast instead of signaling pets one at a time result in more useless wakeups than are necessary?

* Broadcasting a signal creates many unnecessary wakeups because signal only wakes up one thread when broadcast wakes them all. When using broadcast, only one thread can enter the critical section resulting in all the other animals returning to wait.

Meeting the *bounded waiting* criterion for the critical-section problem was not required for this assignment, but it is possible that your solution does. If your implementation satisfies it, explain why. Otherwise, describe what additional variables or constructs that would be needed to implement it and when they would be updated.

* Bounded waiting prevents a process from having to wait forever. My program currently does not implement bounded waiting and can easily have animals wait forever. To fix this, in the pet\_counter struct, an int called ‘age’ could be implemented. In my implementation, a function called controller is called whenever the room is empty, and the controller function decides which animal should enter the room based on what animal type has the most animals waiting. If each animal type had an age, it increments every time the controller function is called. Instead of just letting the animal group in with the most waiting, you would let the animal in with the highest waiting + age. You would need to reset age after an animal is let in. This would prevent an animal from waiting forever.

The rules regarding the wolves provide an example of a classic synchronization problem (one of the ones mentioned in class). Which problem is it? What do the wolves correspond to and why?

* I believe that the wolves are an example of the readers writer’s problem where the wolves are the writers. Multiple writers cannot write at the same time so they must take their turn, just like the wolves. Also, if a process is writing, other processes cannot read which represents the scenario if a wolf is in the playroom, the other animals can’t join.

This assignment initially used the term ``deadlock'' to refer to the situation where a pet is never woken up, even after the room empties. Why specifically was this the wrong term to use?

* Deadlock occurs when every process in the set is waiting for an event that can only be caused by another process in the set. Whenever an animal is waiting, it requires a resource, but the other animals don’t carry any resources with them. This is an implementation problem resulting in starvation and is not a deadlock.

Suppose the pet shop wants to give their pets more time to play. They buy the office next door and create a separate shop identical to the first, with the same rules. They want to be sure that both playrooms are being used as much as possible, so if the pets in Shop A are not currently playing, some pets will be brought from Shop B and vice versa. Describe an analogous issue from the textbook's chapter on scheduling and how it would apply to the pet shop's logistics in this scenario.

* This scenario could be compared to two multiprocessor scheduling and specifically symmetric multiprocessing. This is because both Shop A and Shop B can theoretically have their own scheduling for the room and separate queues. In an actual symmetric multiprocessor, the private ready queue would be comprised of threads instead of animals. The queue could also be a multilevel feedback queue which would feature two separate queues for the shops and animals would be able to move from one queue to the other.

If the two shops were connected instead, so that each pet could choose to wait for Playroom A or Playroom B, would it be possible to reach deadlock? Explain.

* If waiting for a specific playroom was its own queue, there cannot be a deadlock. This is because they have their own individual resources that will not overlap. This is also true because after an animal leaves the room, it does not take any resource with it so the animal would be free to enter the queue for the other playroom without cross contaminating the resources.