Hungry Lizard Crossing

Noah Nickles, Dylan Stephens
COP 4634 Systems & Networks I
10/28/2024

Description

This project demonstrates the synchronization of threads using primitives such as mutex locks and semaphores. Each thread that is represented by a lizard needs to be synchronized in order to prevent the threads represented by cats from "playing" with the lizards if too many attempt to "cross the driveway" at once. The driveway represents the critical section in which multiple lizard threads will start and complete their lifecycle routine before going back to sleep.

Since an outline of the project was given to us to fill in the rest, quite a few changes had to be made in order for the program to function properly. A discussion detailing the changes to the code is below.

Discussion

Two mutexes and one semaphore were added.

- cout_mutex: This is used to ensure debug output does not overlap in the cout stream. As such, any function containing debug output has had a lock_guard applied using this mutex.
- crossing_mutex: This is used so the counter variables "numCrossing…" do not have any race conditions.
- driveway_sem: This is used to control the number of lizards on the driveway at a given time.

The run() and wait() functions for both the Cat and Lizard classes were changed in the following ways:

- run(): First, it checks if the thread accessing this function is not already initialized, then it initializes it with the new keyword.
- wait(): First, it checks if the thread accessing this function is already initialized and it is joinable. If so, it will join the thread to the caller thread.

The functions sago2MonkeyGrassIsSafe() and monkeyGrass2SagoIsSafe() now has a sem_wait() call to make the thread wait for a spot on the driveway.

The functions crossSago2MonkeyGrass() and crossMonkeyGrass2Sago() now has a code block containing the crossing_mutex lock and increments the relevant "numCrossing…" variable. At the end of the function, after the crossing has completed, again it uses a code block with a lock but decrements the relevant "numCrossing…" variable. Both of the functions also now have extra debug printing to show the number of lizards crossing at a time.

The functions madeIt2MonkeyGrass() and madeIt2Sago() now has a sem_post() call to release a spot on the driveway.

The function lizardThread() now has the lifecycle of the lizard as described in the project description added to the while(running) loop.

The main() function was updated to add a vector for Cat objects along with a sem_init() call to initialize the driveway_sem. Additionally, it creates and runs the Cat and Lizard objects/threads, then waits for them to terminate after the WORLDEND time was reached. Subsequently all objects are deleted to clean up the allocated resources along with a sem_destroy() call to clean up the semaphore. Also, a final debug statement was added at the end to print when the world ends.

The unidirectional version of the program changed nearly every comment and the organization of functions and classes for added clarity and readability. Also, logic was added using an enum and condition variable along with a mutex and semaphore to ensure all lizards move in the same direction. As such, comments with initials were omitted as around 80% of the file would be tagged with that.

Results

Table 1 Default Values Results Table (Bidirectional)

WORLDEND (s)	Number of Cats	Maximum Number of Lizards Crossing	Lizards Safe?
30	2	4	Yes
90	2	4	Yes
180	2	4	Yes

Table 2 Doubled Cats and Max Lizards Crossing Results Table (Bidirectional)

WORLDEND (s)	Number of Cats	Maximum Number of Lizards Crossing	Lizards Safe?
30	4	8	Yes
90	4	8	Yes
180	4	8	Yes

Table 3 Default Values Results Table (Unidirectional)

WORLDEND (s)	Number of Cats	Maximum Number of Lizards Crossing	Lizards Safe?
30	2	4	Yes
90	2	4	Yes
180	2	4	Yes

Table 4 Doubled Cats and Max Lizards Crossing Results Table (Unidirectional)

WORLDEND (s)	Number of Cats	Maximum Number of Lizards Crossing	Lizards Safe?
30	4	8	Yes
90	4	8	Yes
180	4	8	Yes

Issues Encountered

When doing the bidirectional version of the project, no issues were encountered. However, doing the extra credit unidirectional version two errors were encountered but solved relatively quickly.

1. The order of operations was wrong. We were calling direction_CV.wait() before sem_wait() which led to deadlocks.
2. The timing of incrementing “numCrossing…” was wrong. In the bidirectional version, this value is increment in the relevant “cross…” function. This led to a collision on the driveway. However, due to it being needed as a constraint for the condition variable, it needed to be moved to the relevant “…IsSafe” function. Originally, it was interpreted that this variable would be incremented at the start of the “cross…” function as that is when the actual crossing happens. However, we were able to re-interpret it as showing the intent to start crossing, leading to solving the issue.