HUNGRY LIZARD CROSSING

OVERVIEW

This assignment will provide experience with multi-threaded applications and the synchronization primitives (mutex lock and semaphore, specifically) that are required for their proper use. You will create a program that synchronizes access between threads to a shared data structure.

THE SCENARIO

My house has a herd of lizards and two cats. The cats sleep most of the time and their favorite toy is a lizard. The lizards live in the sago palm but find food across the driveway in the monkey grass. If too many lizards try to cross the driveway at once, the cats will see them and "play" with them. Your job is to synchronize the lizard crossings to prevent any lizards from becoming cat toys.

THE PROGRAM

Write a program in ANSI C to create *N* threads representing lizards and two additional threads each representing a cat. Each lizard thread will sleep for some random amount of time and wake up hungry. The hungry lizard thread will attempt to cross the driveway in a safe manner as shown in the pseudo code below. Crossing the driveway will take some number of seconds. Once on the other side of the driveway, the lizard will eat for some random amount of time in the monkey grass. After eating, the lizard will return home to the sago palm as soon as it safely can and sleep again. Each of the two cat threads will periodically sleep but when awake it will check out the driveway to see if there are any lizards to play with. If the cat sees too many lizards, it will play with them, causing the entire program to terminate. The two cat threads will not differ in their behavior but they will differ in which one is awake and check on the lizzards.

Use one or more locks and semaphores (do not use monitors) to control access to the driveway (the shared resource). Make sure your implementation follows these rules:

- Do not allow too many lizards to cross the driveway at once.
- Do not use busy waits to control lizards.
- Allow the maximum possible lizards to simultaneously cross.

STARTER CODE & CHANGES

You will be provided with a single file that contains a beginning implementation of the project solution. The file `lizard.c` contains many comments and hints about completing the project. The file can be downloaded through *eLearning*. Make changes and additions to this file, but no deletions. You must not create any other source code files. Functions marked 'Completed' must remain unchanged.

In the source code, make in-line comments with your team member initials to mark the lines of sample code you changed or added. You will not be making any deletions. If you do not include comments in your source code to make it more readable, points will be deducted. Here is an example of how a team with members named Pupil Uno and Lerner Zwei would mark a change:

Original code: `int counter = 1;`

Modified code: `int counter = 0; //PU LZ`

IMPLEMENTATION SUGGESTIONS

Each lizard thread will follow an algorithm similar to the one given below. The algorithm is in pseudo code and **NOT IN C**. Do not attempt to make this code run as-is. However, the names of the functions provided in `lizard.c` will be very familiar when you read the code and comments.

```
while (world has not ended)
        sleep for up to MAX_LIZARD_SLEEP seconds
        wait until [sago -> monkey grass] crossing is safe
        cross [sago -> monkey grass]
                it takes up to CROSS_SECONDS seconds to cross
        eat in the monkey grass
                it takes up to MAX_LIZARD_EAT seconds to eat
        wait until [monkey grass -> sago] crossing is safe
        cross [monkey grass -> sago]
                it takes up to CROSS_SECONDS  seconds to cross
```

Each cat thread will follow an algorithm that entails the action described above. You will see the code of the cat implemented in the starter code. Access to variables shared between the threads must be protected. Lock(s) and semaphore(s) must be properly initialized, used, and destroyed.

EXTRA CREDIT

An extra credit challenge is to prevent bidirectional travel. If one or more lizards are crossing in one direction, other lizards wanting to cross in the opposite direction must wait. This addition is not simple and could take a great deal of time to complete. Your extra credit solution will only be graded if a README.txt file is also submitted indicating that you completed the extra credit portion. The way to toggle between the bidirectional and unidirectional modes is by changing the UNIDIRECTIONAL variable in the provided file `lizard.c`. However, to avoid possible corruption of the bidirectional solution write the code for the unidirectional solution in a separate file named `lizardUni.c` and submit both files as your solution.

ANALYSIS

The report *analysis.pdf* is a PDF-converted document that must contain the following information:

- A short description of the problem and how the code changes solves it.
- A discussion of all the changes made to the code.
- A table with results from multiple runs with different run times (constant WORLDEND in the source code) for the simulation.
- Issues encountered in developing the solution.

TABLE 1 SAMPLE RESULTS TABLE

WORLDEND (s)	Maximum Number of Lizards Crossing	Lizards safe?
30	4	Yes
180	4	Yes

Try to confirm the maximum number of lizards crossing by printing the appropriate counter values as a sum to the screen.

DELIVERABLES

Your project submission should follow the instructions below. Any submissions that do not follow the stated requirements will not be graded.

1. Follow the submission requirements of your instructor as published on *eLearning* under the Content area.
2. You must submit the following files for this assignment:
    a. `lizards.c` (the source code file, UNIDIRECTIONAL set to 0 and WORLDEND to 180)
    b. `lizardsUni.c` (only submit this if you completed the extra credit assignment)
    c. analysis.pdf (the results from an experiment and the changes in the file)
    d. Makefile
    e. README.txt if you completed the extra credit unidirectional option or completed a partial solution of the problem

DUE DATE

The project is due as indicated by the Dropbox for project 3 in *eLearning*. Upload your complete solution to the dropbox and the shared drive. Upload ahead of time, as last minute uploads may fail. Please review the policy in the syllabus regarding late work.

TESTING

Test your code thoroughly on the public servers `ssh.cs.uwf.edu`. We will test your program on the testing server that uses the same OS and programming environment as the public servers.

COMMENTS

The provided code is designed to accept an optional argument on the command line. If the `-d` option is given debugging output statements will be printed during execution. I suggest using the debugging option while developing your program.

GRADING

This project is worth 100 points total. The rubric used for grading is included below. Keep in mind that there will be deductions if your code does not compile, has memory leaks, or is otherwise, poorly documented or organized.

Submission	Perfect	Deficient		
eLearning	5 points individual files have been uploaded	0 points files are missing		
shared drive	5 points individual files have been uploaded	0 points files are missing		
Compilation	**Perfect**	**Good**	**Attempted**	**Deficient**
Makefile	5 points make file works; includes clean rule	3 points missing clean rule	2 points missing rules; doesn't compile project	0 points make file is missing
compilation	10 points no errors	7 points some warnings	3 points some errors	0 points many errors
Documentation & Style	**Perfect**	**Good**	**Attempted**	**Deficient**
documentation & program structure	5 points follows documentation and code structure guidelines	3 points follows mostly documentation and code structure guidelines; minor deviations	2 points some documentation and/or code structure lacks consistency	0 points missing or insufficient documentation and/or code structure is poor; review sample code and guidelines
Threads	**Perfect**	**Good**	**Attempted**	**Deficient**
creates threads with pthread_create	10 points correct, completed	7 points minor errors	3 points incomplete	0 points missing
synchronizes threads to limit access to shared resource (driveway)	15 points correct, completed	11 points minor errors	4 points incomplete	0 points missing
avoids race conditions	15 points correct, completed	11 points minor errors	4 points incomplete	0 points missing
joins with all threads at the end of execution	10 points correct, completed	7 points minor errors	3 points incomplete	0 points missing
Results, Changes, Issues	**Perfect**	**Good**	**Attempted**	**Deficient**
problem description & results from multiple run times	8 points correct, completed	6 points minor errors	2 points incomplete	0 points missing
discussion of changes	6 points correct completed	4 points minor errors	2 points incomplete	0 points missing
discussion of issues encountered	6 points correct, completed	4 points minor errors	2 points incomplete	0 points missing

I will evaluate your solution as attempted or insufficient if your code does not compile. This means, if you submit your solution according to my instructions, document and structure your code properly, provide a makefile and a report but the submitted code does not compile or crashes immediately you can expect at most 64 out of 100 points. So be sure your code compiles and executes properly.