4. Programming Exercises

4.1 Solving a Synchronization Problem using Condition Variables

This project offers you an opportunity to write a straightforward concurrent program and to get a more detailed understanding of how to use threads to solve synchronization problems. We have provided you with basic driver code that starts a predefined number of threads. You are responsible for what those threads do.

Again, remember to specify a seed to use in the random number generator by editing your `sys161.conf` file. It is a lot easier to debug initial problems when the sequence of execution and context switches is reproducible.

When you configure your kernel for ASST1, the driver code and extra menu options for executing your solutions are automatically compiled in.

You must implement a solution for the casts-and-mice problem using the *CV* mechanism developed in project 3. You should start your implementation by modifying the existing source code file named `catlock.c`, which is located in the `src/kern/asst1` directory.

4.2 The Synchronization Problem: Cats and Mice

A professor has several cats and some mice that like to hang out in her house. The cats and mice operate need to work out a deal where the mice are allowed to steal bits of cat food, provided that the cats never see the mice actually doing so. If a cat sees a mouse eating from a cat dish, then the cat is obligated to eat the mouse.

You must implement the eating habits of cats and mice, which follow the synchronization policy below.

- no mouse ever gets eaten, and
- neither the cats or the mice starve.

Important! You must implement a solution (i.e., `catlock.c`) using the lock mechanism coupled with condition variables. Each cat and mouse thread should identify itself and which dish it is eating from at the point when it begins and when it finishes eating. Simulate a cat or mouse eating by calling `clocksleep()`.

We make the following assumptions while solving this synchronization problem:

- there are two cat food dishes, 6 cats, and two house mice to be coordinated,
- only one mouse or cat may eat at a given dish at any one time,
- if a cat is eating at either dish, a mouse attempting to eat from the other dish will be seen and therefore eaten,
- when cats aren't eating, they will not see mice eating.

The driver code for the Cats-and-Mice problem is found in the following existing source-code file, the original version of which only forks the required number of cat and mouse threads.

`kern/asst1/catlock.c`

Important! There is no necessity of modifying `kern/asst1/catsem.c`, which is reserved for another solution using the semaphore mechanism.

4.3 Written Exercises

Once you have completed your solution, answer the following questions in `exercises.txt` file.

- Explain how to avoid starvation in your implementation.
- Comment on your experience of implementing the Cats-and-Mice program. Can you derive any principles about the use of the lock and condition variable synchronization primitives?

5. Deliverables

Make sure the final versions of all your changes are added and committed to the `Git` repository before proceeding. We assume that you haven't used `asst1b-end` to tag your repository. In case you have used `asst1b-end` as a tag, then you will need to use a unique tag in this part.

```
%cd ~/cs161/src
$git add .
%git commit -m "ASST1b final commit"
$git tag asst1b-end
%git diff asst1b-start..asst1b-end > ../project4/asst1b.diff
```

`asst1b.diff` should be in the `~/cs161/project4` directory. It is prudent to carefully inspect your `asst1b.diff` file to make sure that all your changes are present before compressing and submitting this file through `Canvas`. It is your responsibility to know how to correctly use `Git` as a version control system.

Important! Before creating a tarball for your project 4, please ensure that you have the following two files in the `~/cs161/project4` directory.

`exercises.txt` and
`asst1b.diff`

You can create a tarball using the commands below:

```
%cd ~/cs161
%tar vfzc project4.tgz project4
```

Now, submit your tarred and compressed file named `project4.tgz` through Canvas. You must submit your single compressed file through Canvas (no e-mail submission is accepted).

6. Grading Criteria

1) A lock-CV based solution in `catlock.c`: 70%
2) Adhering to coding style and documentation guidelines: 10%.
3) Written exercises (`exercises.txt`): 20%.

8. Make Your Code Readable

It is very important for you to write well-documented and readable code in this programming assignment. The reason for making your code clear and readable is two-fold. First, there is a likelihood that you will read and understand code written by yourselves in the future. Second, it will be a whole lot easier for the TA to grade your programming projects if you provide well-commented code.