CS105 – Computer Systems Fall 2023

Assignment 8: Sync Lab

Due: Tuesday, November 21, 2023 at 11:59pm

In this lab, you will be synchronizing a multi-threaded program. As usual, the starter code is available as a tar file `synclab.tar` that can be downloaded from the course website or can be found in the `/data` directory on the course vm. As always, please complete this assignment in pairs.

To get started, unpack the tar file using the command "`tar xvf synclab.tar`", and look inside. You should see three files: `cv.c`, `cv.py`, `Makefile`. Yes, this assignment may completed in either C or in Python! The C files and the Python files contain equivalent starter code in the two language. Before you go any further, you should decide which language you would like to use to complete the assignment. If you choose to solve the assignment in C, you can use the Makefile to compile your code.

When you open up the `cv` file in the language of your choice, you will find that it contains code that implements a club patronized by two types of people: goths and hipsters. Goths and hipsters are both implemented as threads; each of these threads enters the club, hangs out for a couple of seconds, and then leaves the club. Your job will be to add synchronization to these programs to enforce various constraints using locks and condition variables.

Your Tasks

For this lab, you should complete the following three tasks.

Task 1: Your first is to add synchronization that ensures that:

1. The club is always exclusively Goth or Hipster, i.e., no Goth should enter as long as there are Hipsters in the club, and vice versa.
2. The club should always be used as long as there are customers.

If you are solving this assignment in Python, you should only modify the methods of the Club class to solve this assignment. You will need to modify `__init__`, `goth_enter`, `goth_exit`, `hipster_enter`, and `hipster_exit`.

If you are solving this assignment in C, you will need to modify the struct definition at the top as well as the functions `club_init`, `goth_enter`, `goth_exit`, `hipster_enter`, and `hipster_exit`.

Task 2: Once you have successfully solved the first task, you might observe that your solution suffers from a problem known as *starvation*: there is no guarantee that a thread waiting to enter the club will eventually be allowed to enter the club. For example, the club might become Goth and remain exclusively Goth for all time, leaving the waiting Hipsters to wait at the door, talking about how cool the club was when it was still underground. Modify your synchronization so that it is *starvation-free*, that is all threads are guaranteed to eventually make progress.

Task 3: You might also observe that your code doesn't enforce any capacity constraints for the club. In the starter code, the capacity of the club is set equal to the total number of people (Goths plus Hipsters), so this isn't a problem, but you could run into trouble if you increase the number of people (i.e., threads) without increasing the capacity of the club. Modify your synchronization to explicitly enforce the restriction that the number of people in the club cannot be greater than the capacity of the club.

Testing

Testing and debugging synchronization code is notoriously difficult. My best advice is to leave your code running for a while and see whether the sanitycheck function detects any problems. I would also recommend that you try varying the number of Goths and the number of Hipsters (but make sure the club capacity is always big enough to accommodate all the people if you have not yet implemented Task 3).

Feedback

Create a file called `feedback.txt` that answers the following questions:

1. How long did each of you spend on this assignment?
2. Any comments on this assignment?

As always, how you answer these questions **will not affect your grade**, but whether you answer them will.

Submission

Submit your `cv.*` file and your feedback file on Gradescope. Remember to include all team members as collaborators when you submit! Also, be sure the names of all team members are *clearly* and *prominently* documented in the comments at the top of both submitted files.

Language-Specific Notes

C

In class, I mentioned that there are convenient initialization macros for locks and condition variables. Unfortunately, these can only be used if you initialize the synchronization primitives when they are declared. Since this assignment declares locks and condition variables inside a struct and initializes them with the `club_init` function later, you will instead need to use the underlying initialization functions. For example, if your club struct has a field of type `pthread_mutex_t` named `lock` and a field of type `pthread_cond_t` named `cv`, you would initialize these fields using the following function calls:

```
pthread_mutex_init(&(club->lock), NULL);
pthread_cond_init(&(club->cv), NULL);
```

Python

The starter code assumes that you are running Python 3. Note this this version is not compatible with Python 2. Plan accordingly. (You can run Python3 on the course VM using the command `python3` if you don't have a copy conveniently on your local machine.)