Neil Kanakia October 28, 2022 CS361: Midterm

Question 1:

a) 6 reflections

b) 4 assignments

c) I have attended all live lectures and watched others asynchronously

d) Was confident about most homework answers when I submitted them

Question 2

a) HW4 was the most insightful since it gave me the best understanding of how much of an improvement threads can offer.

b) Even though none of them were too bad HW3 was frustrating since even though most of the code was provided, I found it hard to debug my code. (It would have been better if HW4 was before HW3 since I would be more equipped to deal with the debugging aspect)

c) If I had to resubmit my HWs it would be HW3 since I would want to make my code more robust and add colors to the set.

Question 3

a) I am taking this class as an elective. I wanted to have a better understanding of c++ and spoke to who is now the the TA, Adeeb who recommended it to me.

b)

Question 4

This threading puzzle is an example of the consumer producer problem.

In this case we will assume that:

- The pledge is the producer
- The communal keg is the buffer
- The partiers are the consumers

We can also assume semaphors:

- A keg lock. Which will be a mutex
- A keg indicator that shows how many drinks are available

In our case the buffer is finite. Which means that the consumer(pledge) will have to wait for the consumers to consume.

At a time the keg can hold N number of servings.

When N = 0, the pledge is forced to wake up and replace the keg (buffer).

```
keg = buffer.get()
function getBeer(pledge PledgeName)
{
    pledge.getbeer()
}

// Producer section
while(beer is needed)
{
    pledge john = new Pledge();
    keg = getBeer(john)
    mutex.wait() // Pledge ensures that the keg is open
    keg.get(beer) // Beer is in the communal keg
    mutex.signal() // The communal keg is unlocked
    items.signal() // Tell partiers that beer exists
}
```

This could lead to a deadlock where the keg is empty and the partier wants to drink but will need to wait

Since the pledge has gone to bring a new keg therefore the partier will wait forever

Therefore a better solution would be to have a communal keg that is made up of multiple kegs, which is implemented as a queue.

Now the pledge can bring multiple kegs and stack them in line before the next one is empty.

This solution is more optimized since the consuming and producing is decoupled

Question 5

Grade: A- I am giving this grade based on my understanding of concurrency based on this class.

I started with little to no knowledge of c++ or threads.

Through the assignments and various readings, I have a little more understanding. Even though it still feels like it is abstract.

I have a complete understanding of mutex locks and semaphore. Even though my understanding of c++ is not at a high level (which I am interested in working

on)

Looking at my previous homework grades, it seems like I lost most points in c++ errors which I would like to improve.

HW4, was the one where I perfected my code to make sure I was utilizing c++ correctly.

The textbook c++ Concurrency in Action was a great resource to help me do this.

It provided a high level understanding of concepts and provides great analogies to simplify these concepts.

The things that I am lacking in: - Understanding of OS on a deeper level which would have greatly helped. - Registers, program counters. - Under the hood understanding of the library. It would be great if we could look into this at some level to know what's really happening when we invoke/ kill these threads and how a compiler handles them.