There are many choices for HW, just have some practice writing code, we will not review

1. You can take your TicTacToe game and set the computer’s turn to happen in a new thread so you can delay the computer’s turn.
2. You can time how long the game took and write a high score file to track the top 5 fastest wins. You can show the scores in the game.

These next two, just try to code something to address these problems. I don’t expect you to solve them. I myself have not written any code for it

1. Write code to deal with the **producer-consumer problem**. The problem describes two processes, the producer and the consumer, which share a common collection, a fixed-size buffer used as a queue.

The producer’s job is to generate data, put it into the buffer, and start again.

At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.

Problem:

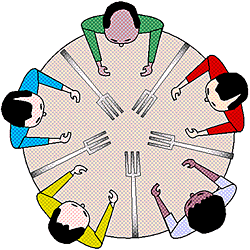
To make sure that the producer won’t try to add data into the buffer if it’s full and that the consumer won’t try to remove data from an empty buffer.

Solution

The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

An inadequate solution could result in a deadlock where both processes are waiting to be awakened.

1. Write code to deal with the **dining philosopher's problem**. It is the classical problem of synchronization which says that Five philosophers are sitting around a circular table and their job is to think and eat alternatively. A bowl of noodles is placed at the center of the table along with five chopsticks for each of the philosophers. To eat a philosopher needs both their right and a left chopstick. A philosopher can only eat if both immediate left and right chopsticks of the philosopher is available. In case if both immediate left and right chopsticks of the philosopher are not available then the philosopher puts down their (either left or right) chopstick and starts thinking again.



The dining philosopher demonstrates a large class of concurrency control problems hence it's a classic synchronization problem.

The goal is to come up with a procedure that the philosophers achieve their goal of eating and thinking without getting starved to death. Following pseudo code is the procedure to follow when solving this problem.

Each philosopher should:

*while(true){*

*think(); //Initially thinking about the whole universal things.*

*pick\_up\_left\_fork(); //Readying to eat as philosopher gets hungry eventually.*

*pick\_up\_right\_fork();*

*eat();*

*put\_down\_right\_fork();*

*put\_down\_left\_fork();*

*think(); //Back to start thinking as hungry is over.*

*}*

One of the issues is that any given fork needs to be shared with the philosopher’s neighbor and he can’t eat until he has the two forks on either side of himself.