Bounded Buffer

Mission

To understand and to implement a bounded Buffer, a thread safe exchange of data

Background

C#Note Prog04 pp.1-21, slides Concurrency-part1.pdf & Concurrency-part2.pdf
Example see: Producer–consumer problem - Wikipedia

Set up scenario

Exercise 1 – Set up an experiment Producer - Consumer

You are to design and implement one or more subsystems to produces 'items' (of some kinds). These items have to be sent/given to another subsystems which consumes the items and print them out.

Step 1: Implement a model class 'Item' with the properties **Id**, **Value**(int), make sure the class itself generate a unique id.

Step 2: Create a class 'Experiment'

Step 3: In the Experiment class design and implement a method 'Producer', which generate a new item with random values and put them into a queue to the consumer. The generation of items should be with random time periods e.g. between 10-150 msec.
(Hints use Thread.Sleep and the class Random)

Step 4: In the Experiment class design and implement another method 'Consumer', which takes out Items from the queue and prints them out. Again taking items out of the queue should be with random time periods.

Step 5: In the Experiment class design and implement a method 'Start()', to instantiate a queue object, together with 4 producers one in each thread and two consumers.

Step 6: In program create an object of the Experiment class and call the method Start().
Notice what happened?

Next step is to implement a Bounded Buffer class

Exercise 2 – Design and implement a BoundedBuffer

You are to implement a new class BoundedBuffer, preferable generic, to exchange data thread safe between threads

Step 1: Create a class BoundedBuffer

Step 2: Make four instance fields:

- a Queue ('buffer').
- one semaphore ('empty') to indicate the queue is empty.
- one semaphore ('full') to indicate the queue is full.
- a lock ('lockObj') for atomize / synchronize thread access.

Step 3: Make a method Insert (item)

- Use the lock to ensure only one thread at the time can execute the method
- Use the full-semaphore to ensure free space in the queue
- insert the item in the queue
- Release the empty-semaphore

Step 4: Make a method Take () – return an item

- Use the lock to ensure only one thread at the time can execute the method
- Use the empty-semaphore to ensure the queue do contains at least one item
- take the item from the queue
- Release the full-semaphore

Excercise 3 – Use your BoundedBuffer in your experiment from exercise 1

In the program from exercise 1 refactor the codse to use your bounded buffer instead of a plain queue.
What happen now when running?
What happen if you start 10 producers ? or 10 consumers?

Extra E1: In C# exists a data structure `ConcurrentQueue`, why is this data structure not direct a substitute for the BoundedBuffer?