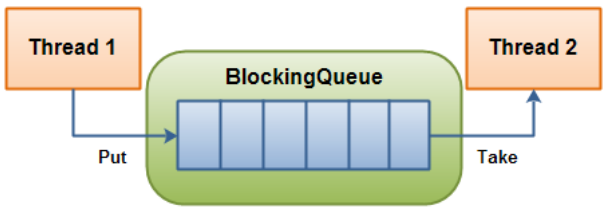
Blocking Queues

Jakob Jenkov
Last update: 2014-06-23

A blocking queue is a queue that blocks when you try to dequeue from it and the queue is empty you try to enqueue items to it and the queue is already full. A thread trying to dequeue from an e queue is blocked until some other thread inserts an item into the queue. A thread trying to enque item in a full queue is blocked until some other thread makes space in the queue, either by dequ one or more items or clearing the queue completely.

Here is a diagram showing two threads cooperating via a blocking queue:



A BlockingQueue with one thread putting into it, and another thread taking from it

Java 5 comes with blocking queue implementations in the `java.util.concurrent` package. You read about that class in my **java.util.concurrent.BlockingQueue** tutorial. Even if Java 5 comes blocking queue implementation, it can be useful to know the theory behind their implementation

Blocking Queue Implementation

The implementation of a blocking queue looks similar to a **Bounded Semaphore**. Here is a simp implementation of a blocking queue:

```
public class BlockingQueue {

  private List queue = new LinkedList();
  private int  limit = 10;

  public BlockingQueue(int limit){
    this.limit = limit;
  }


  public synchronized void enqueue(Object item)
  throws InterruptedException  {
    while(this.queue.size() == this.limit) {
      wait();
    }
    if(this.queue.size() == 0) {
      notifyAll();
    }
    this.queue.add(item);
  }


  public synchronized Object dequeue()
  throws InterruptedException{
    while(this.queue.size() == 0){
      wait();
    }
    if(this.queue.size() == this.limit){
      notifyAll();
    }

    return this.queue.remove(0);
  }

}
```

Notice how `notifyAll()` is only called from `enqueue()` and `dequeue()` if the queue size is equal size bounds (0 or limit). If the queue size is not equal to either bound when `enqueue()` or `dequeu` called, there can be no threads waiting to either enqueue or dequeue items.

Next: **Thread Pools**

Jakob Jenkov

in a deadlock where the program is waiting for a task to be written to queue, but no task can be written as the lock is held by the "waiting" process?

Reply • Share ›

Jakob Jenkov Mod → pymd • a year ago

The call to object.wait() releases the lock on the object (the queue) so other threads can call the enqueue() method.

Reply • Share ›

planepotdisqus • a year ago

```
if(this.queue.size() == this.limit){
notifyAll();
}
```

This wakes up a waiting enqueue thread. If the element has been dequeued, all is good, but if the thread wakes up before the element is removed, it queue is still at limit.

All Trails | Trail TOC | Page TOC | Previous | Next

Then waits() again.

Reply • Share ›

Jakob Jenkov Mod → planepotdisqus • a year ago

The enqueing thread cannot exit the wait() call until the dequeuing thread leaves the dequeue() method. Thus, the enqueuing (waiting) thread able to execute the while loop until the element is actually dequeued. Awakening a thread does not give it the lock of the object. It just wakes makes it ready to enter the synchronized block when possible.

Reply • Share ›

Adrian • a year ago

Oops. Forgot that it must be while not if. I stand corrected. Thank you for pointing that out.

Reply • Share ›

Adrian • a year ago

This example is great and I have been playing with it. However, I think you should add a check on queue first if its empty before removing element a because another thread might have emptied the queue already. I think the return statement of dequeue method should be return (queue.isEmpty()) ? queue.remove(0).

Reply • Share ›

Jakob Jenkov Mod → Adrian • a year ago

The dequeue() is synchronized, so once a thread leaves the

while(this.queue.size() == 0) { ... }

There is at least 1 element in the queue, and no other thread can take it (because the thread owns the lock on the dequeue() method).

Therefore it is not necessary with another check to see if the queue is empty.

1 Reply • Share ›