Comments on AuctionServer Synchronization

September 2016

The way you use synchronization in your code depends on the invariants. Those can be extracted, more or less uniquely, from the description of the problem.

Additionally, the use of synchronization also depends on the structuring of your code. For instance, if you have an invariant in which variables A and B are involved, and your code spreads the access to those variables throughout a whole method, synchronizing the whole method on `this` using `synchronized` might not be a bad option. However, your implementation may result in slower execution, which is not desirable.

You can improve the performance if you use multiple locks for blocks of code. However, you still need to ensure that the access to the variables that are involved in the same invariants is synchronized by the same lock.

For your Homework 3/4, the following table shows the variables that need to be protected and the methods they are used in. While the fact that two variables are used in the same method does not necessarily imply they are dependent, it still is a strong indication of such a situation.

Variable	**Method**	**Invoked by**
soldItemCount	checkBidStatus()	Bidder
revenue	checkBidStatus()	Bidder
maxBidCount	submitBid()	Bidder
maxSellerItems	submitItem()	Seller
serverCapacity	submitItem()	Seller

itemsUpForBidding	submitItem()
	submitBid()
		Seller
		Bidder

lastListingID	submitItem()	Seller

itemsAndIDs	submitItem()
	submitBid()
	checkBidStatus()
	itemPrice()
		Seller
		Bidder
		Bidder
		Bidder

highestBids	submitBid()
	checkBidStatus()
	itemPrice()
	itemUnbid()
		Bidder
		Bidder
		Bidder
		------ 	

highestBidders	submitBid()
	checkBidStatus()
		Bidder
		Bidder

itemsPerSeller	submitItem()	Seller
itemsPerBuyer	submitBid()	Bidder

The next table lists the methods and shows which of the methods use the variables that need protection. It also shows (in the right most column), in which other methods the same variables are used. This tells you that if the multiple methods use the same variable that needs protection, the methods should be synchronized on the same lock.

Method	**Variables used**	In other methods
submitItem()	maxSellerItems serverCapacity itemsUpForBidding (D) lastListingID itemsAndIDs (A) itemsPerSeller	0 0 submitBid() 0 submitBid(), checkBidStatus(), itemPrice() 0

submitBid()	maxBidCount
	itemsUpForBidding (D)
	itemsAndIDs (A)
	highestBids (B)
	highestBidders (C)
	itemsPerBuyer
		0
		submitItem()
		submitItem(), checkBidStatus(), itemPrice()
		checkBidStatus(), itemPrice()
		checkBidStatus()
		0

checkBidStatus()	soldItemCount
	revenue
	itemsAndIDs (A)
	highestBids (B)
	highestBidders (C)
		0
		0
		submitItem(), submitBid(), itemPrice()
		submitBid(), itemPrice()
		submitBid()

itemPrice()	itemsAndIDs (A)
	highestBids (B)
		submitItem(), submitBid(), checkBidStatus()
		submitBid(), checkBidStatus()

Consequently, this table shows you that the code in which `itemsAndIDs` is accessed should be synchronized on the same lock. In other words, this variable is shared among all the four methods.

The rest of the synchronization strategy strongly depends on the structure of your code, as mentioned at the beginning of this post. But the main concern here is – which of the other variables can be accessed synchronously with `itemsAndIDs`?

Clearly, it is safe to use just one lock for all the synchronization. However, this may impose penalty on the speed of your program. The methods `submitBid()` and `checkBidStatus()` must involve the synchronization of large parts of their code on the same lock. However, code segments in which other variables are accessed, those with zeroes in the right most column, could be separated out and controlled by other locks.