Thread Synchronization - how to synchronize access to resources in multithreaded applications - Questions and Answers

Atomic operations.

1. What is an atomic operation?

Unit of work which is guaranteed to not being interrupted.

2. Explain the difference between Exchange() and CompareExchange() methods.

A: CompareExchange() has the third argument - comparand. If the actual value of updated variable is not equal to comparand, CompareExchange() does not update it.

Synchronization events

3. What is the difference between AutoResetEvent and ManualResetEvent?

A: Waiting thread should not call Reset() on AutoResetEvent if it was unlocked by WaitOne(), as event is reset automatically.

4. How can user wait for one or all events?

A: Use WaitAny() and WaitAll() static methods of WaitHandle class.

Locks

5. Is thread synchronization required when app is running on single-processor single core PC? Why?

A: Yes, since preemptive multitasking may interrupt one thread at any time.

6. Is block after lock keyword atomic?

A: No. Lock only ensures that no other thread can enter locked block while current thread is here. Thread which is in locked block can still be interrupted.

7. When should one use Monitor class and when lock keyword?

A: Prefer lock over Monitor where it is possible. However, it is not always possible, i.e. Monitor has TryEnter(timeout) method.

8. Can we lock on a value type? Why?

A: Value types are boxed as lock expects object. Each boxing produces new reference, so locking is does not work.

9. When should one use Mutex or Monitor (lock)?

A: Using Monitors is preferred over Mutex since Mutex is a wrapper to Win32 construct, and Monitor is .NET construct. Though Mutexes are more powerful.

10. What is Semaphore class? When should it be used?

A: Semaphore limits the number of threads that can access a resource or pool of resources concurrently. Usage example - when user wants to limit number of simultaneous DB connections.

11. Imagine the situation: Three threads are reading data protected with ReadWriterLock. Fourth thread is going to update the data. Describe next steps one by one.

A: ReadWriterLock locks all subsequent requests, waits until current read locks are released, then enters write lock block, exits and unlocks pending read requests.

Collections and deadlocks

12. List build-in .NET thread-safe collections

A: ConcurrentDictionary<TKey,TValue>, ConcurrentQueue<T>, ConcurrentStack<T>

13. What is SyncRoot property? Where it exists and when used?

A: SyncRoot is member of collections used as object for lock keyword.

14. How to avoid deadlocks? Look provided code example.

```
static object object1 = new object();
static object object2 = new object();

public static void ObliviousFunction()
{
    lock (object1)
    {
        Thread.Sleep(1000); // Wait for the blind to lead
        lock (object2)
        {
        }
    }
}

public static void BlindFunction()
{
    lock (object2)
    {
        Thread.Sleep(1000); // Wait for oblivion
        lock (object1)
```

```
            {
            }
    }
}
```

A: Preventing deadlocks is large topic, but general rules are:

- Don’t take the fork until you have put the spoon
- Don’t take the fork and the spoon simultaneously

- List When to use a thread-safe collections?

https://docs.microsoft.com/en-us/dotnet/standard/collections/thread-safe/when-to-use-a-thread-safe-collection