

Examination #3

Name: _____ **Instructor's Solution** _____ SUID: _____

This is a closed book examination. Please place all your books on the floor beside you. You may keep one page of notes on your desktop in addition to this exam package. All examinations will be collected promptly at the end of the class period. Please be prepared to quickly hand in your examination at that time.

If you have any questions, please do not leave your seat. Raise your hand and I will come to your desk to discuss your question. I will answer all questions about the meaning of the wording of any question. I may choose not to answer other questions.

You will find it helpful to review all questions before beginning. All questions are given equal weight for grading, but not all questions have the same difficulty. Therefore, it is very much to your advantage to answer first those questions you believe to be easiest.

1. Write all the code for a wrapper that provides thread-safe access for read and write for any type that is or can be defined in C#. Can your wrapper be a drop-in replacement for the underlying type without changing any of the surrounding code?

Answer:

Defining a wrapper for value types is relatively straightforward, and is close to a drop-in replacement. That can't be done as easily for reference types and isn't going to behave like a drop-in.

```
class ValueWrapper<T> where T : new()
{
    T t_ = new T();
    object locker_ = new object();

    public T Contents
    {
        get
        {
            lock (locker_) { return t_; } // returns copy of value
        }
        set
        {
            lock (locker_) { t_ = value; } // replaces held value
        }
    }
}

class ReferenceWrapper<T> where T : new()
{
    object locker_ = new object();
    T t_ = new T();

    public T Contents
    {
        get
        {
            Monitor.Enter(locker_); // return locked
            return t_;
        }
        set
        {
            Monitor.Enter(locker_);
            t_ = value;
            Monitor.Exit(locker_);
        }
    }
    public void Release()
    {
        Monitor.Exit(locker_); // when done unlock
    }
}
```

2. Suppose you want to track access to any element in the Code Repository of Project #5¹ by userid². How would you do that³? Discuss the performance implications of your concept. Are there security implications for this functionality?

Answer:

Note: we are not trying to **control** access; we are trying to **track** access.

Since the number of users will be fairly small, a few hundred at most, and the number of elements is likely to be large, perhaps hundreds of thousands, it would make sense to use the elements as a key in a hashtable with values being a list of userids that have accessed the element. If we don't need to know how many times a user accessed each element we can use a compressed list that does not store duplicate entries. This makes it easy to answer queries like "who accessed element e?".

We may also have to answer queries like "which elements did John Doe access?". For these queries we would use a hashtable with userids as keys with a list of elements accessed as its value. It wouldn't be too expensive to keep both tables and update them in real time as elements are accessed. With these two tables we can do either query in constant time.

We would, of course, have to make the tables persistent, perhaps by writing out to an XML file when the Repository shuts down and reading back when the Repository is started. We would want to do this on a background thread so the Repository comes up quickly.

Many security attacks come from insiders, e.g., authenticated users. We could easily build a tool to scan the second table to see if John Doe was accessing many elements outside his development area. If so, we might want to have a talk with John.

¹ The Project #5 statement is attached.

² This means that for each element in the repository we can say who accessed the element at what date and time.

³ Try to be brief but specific.

3. Show three distinct ways of starting a thread without using the thread class directly.

Answer:

```
class ImplicitThreads
{
    public static void Handler(string msg)
    {
        Console.WriteLine("\n {0}, Thread id = {1}", msg, Thread.CurrentThread.ManagedThreadId);
        Thread.Sleep(200);
    }
    public static async Task HandlerAsync(string msg)
    {
        await Task.Run(() => Handler(msg));
        Console.WriteLine("\n {0}, Thread id = {1}", msg, Thread.CurrentThread.ManagedThreadId);
    }
    static void Main(string[] args)
    {
        Console.WriteLine("\n Main Thread id = {0}", Thread.CurrentThread.ManagedThreadId);

        // implicit thread start #1 - async await

        string arg1 = "Hello CSE681 World using async await";
        Task t2 = ImplicitThreads.HandlerAsync(arg1);

        // implicit thread start #2 - Task.Run

        string arg2 = "Hello CSE681 World using Task.Run";
        Task t1 = Task.Run(() => ImplicitThreads.Handler(arg2));

        // implicit thread start #3 - async lambda

        string arg3 = "Hello CSE681 World using async lambda";
        Func<Task> lambda = async () =>
        {
            await Task.Run(() => ImplicitThreads.Handler(arg3));
            Console.WriteLine("\n {0}, Thread id = {1}", arg3,
                Thread.CurrentThread.ManagedThreadId);
        };
        Task t3 = lambda();

        // implicit thread start #4 - BeginInvoke

        Action<string> act = new Action<string>(ImplicitThreads.Handler);
        string arg4 = "Hello CSE681 World using BeginInvoke";
        IAsyncResult asyncResult = act.BeginInvoke(arg4, null, null);
        act.EndInvoke(asyncResult);
        Task.WaitAll(t1, t2, t3);

        Console.WriteLine("\n\n");
    }
}
```

4. What is the purpose of the `IDisposable.Dispose()` method? When would you write one? When would you use one?

Answer:

`Dispose` allows the user of a `Disposable` class to ask for cleanup of native resources, e.g., file handles, threads, mutexes and semaphores, database locks, etc. If the disposable class is using .net wrapper classes for native resources like `System.IO.File` then those get cleaned-up by calling `Dispose()` in the `Disposable` class. Actually `File.Close()` calls `File.Dispose()`.

You write the `Dispose` method when your class derives from `IDisposable`. You want that whenever your class uses native resources directly or wrapped in .Net classes, e.g., frequently. If your class uses them directly it must release or close or delete them (depending on what you are using). If your class uses an instance of a .Net type that implements `IDisposable` you call `dispose` on the instance.

You call `Dispose()` on a disposable class whenever you have finished using it. Thus cleanup doesn't have to wait for garbage collection.

5. What are your obligations when you write the Operational Concept Document for Project#4⁴?

Answer:

You are obligated to provide an Executive Summary, discuss uses and users, impact on design of satisfying their needs, top-level software partitions with their responsibilities, and critical issues. A suitable partitioning for project #4 is shown in the answer to question #7.

Here are some critical issues for Project #4:

- Issue #1: Large number of stored documents
- Solution: Use categories for server storage and in queries

- Issue #2: Query result size may be large, even with categories
- Solution: For matches > Max show number of matches in each category and allow client to pick a category to show details

- Issue:#3 Search time may be large for some queries
- Solution: Show matches as they are found (sequence of reply messages) and run query searches in parallel

- Issue:#4 Will have some documents with no parents
- Solution: Make their categories also serve as parents so they will be visible in navigation view

- Issue:#5 Occasionally two concurrent clients may attempt to query same file at same time
- Solution: Use file locking as shown in MT2-Q3

- Issue:#6 Need to easily access frequently used documents
- Solution: Create book marks stored in a favorites container so we don't have to repeatedly query large data sets for same results

- Issue: #7 Need to chunk large documents for transfer between client and server
- Solution: Handle chunking in Sender and Service object so chunks from one client are never mixed with chunks from another. Can also post chunks as messages and sort out in a FileHandlerCommunicator.

- Issue: #8 Two clients attempt to check-in different files with same name
- Solution: Use owner name as prefix for all documents, e.g., JFawcett.Parser

⁴ The Project#4 statement is attached at the back of this packet.

6. Describe the instancing models for Windows Communication Foundation. Why do we need them? When would you use each?

Answer:

We have discussed three instancing modes: PerCall, PerSession, and Single.

PerCall creates a service object instance for each service call and discards the object after the call finishes. Each client gets a service object running on a thread dedicated to that client.

PerSession creates a service object if one does not exist that has a leased lifetime. Each time a call is made the lease is extended. If the lease expires because there have been no calls during the last lease the object will be discarded. Each client gets a service object running on a thread dedicated to that client.

Single behaves just like PerSession with the exception that all clients share the same object, so its static and member data have to be synchronized.

We need instancing policies because the server cannot know when a client has finished all use of a service object, and so can not discard them. Without an invocation policy the server memory would fill up with no-longer used objects, causing poor performance and eventually service faults.

Use PerCall when the time between client calls may be long and state does not have to be shared from invocation to invocation.

Use PerSession when state must persist between invocations and also when service objects are large and expensive to construct.

Use Single when state needs to be shared between clients and the performance degradation due to locking is acceptable.

7. Draw a package diagram for your planned implementation of Project #4. Describe the responsibilities and interactions of each package.

Answer:

- Executives create and start client and server objects
- Handlers process messages, prepare, and send replies
- Dispatchers route messages to the destinations cited in the message
- Receiver extracts messages from service and sends to dispatcher
- Sender posts messages to the CommService
- CommService implements an Http Service for communicating messages between processes and machines
- Not shown – parent mapping, client local browsing, file up and download, check-in, message handling, categories

