Messaging through memory-mapped files in .NET C#

NOVEMBER 18, 2015 LEAVE A COMMENT (HTTPS://DOTNETCODR.COM/2015/11/18/MESSAGING-THROUGH-MEMORY-MAPPED-FILES-IN-NET-C/#RESPOND)

We saw in this (https://dotnetcodr.com/2015/11/13/writing-to-a-file-using-a-memorymappedfile-in-c-net/) and this (https://dotnetcodr.com/2015/11/17/reading-from-a-memory-mapped-file-in-c-net/) posts how to use memory-mapped files to map an existing file to a memory location that multiple processes had access to on the same machine.

The same key objects, i.e. MemoryMappedFile and MemoryMappedViewAccessor can be used for interprocess messaging purposes. The following code shows how a “server” can create a new shared file mapped to memory. Here we use the CreateNew method for this purpose and give the file a mapping name. Note that this is only an in-memory file, it won’t be saved on disk:

```
static void Main(string[] args)
{
    using (MemoryMappedFile memoryMappedFile = MemoryMappedFile.CreateNew("news-channel", 10000))
    {
        using (MemoryMappedViewAccessor viewAccessor = memoryMappedFile.CreateViewAccessor())
        {
            byte[] textBytes = Encoding.UTF8.GetBytes("The world is going down.");
            viewAccessor.WriteArray(0, textBytes, 0, textBytes.Length);
        }

        Thread.Sleep(100000);
    }

    Console.WriteLine("Main done...");
    Console.ReadKey();
}
```

Consult the links provided above for a full explanation of the code, I won’t repeat it here. You can put the above code into a C# console application and call it Server or something like that. Note that I block code execution for 100 seconds. Without the call to Thread.Sleep the server will simply write to the memory mapped file and then dispose of the MemoryMappedFile object after the “using” block. In which case the client won’t have a chance to connect to the same memory location:

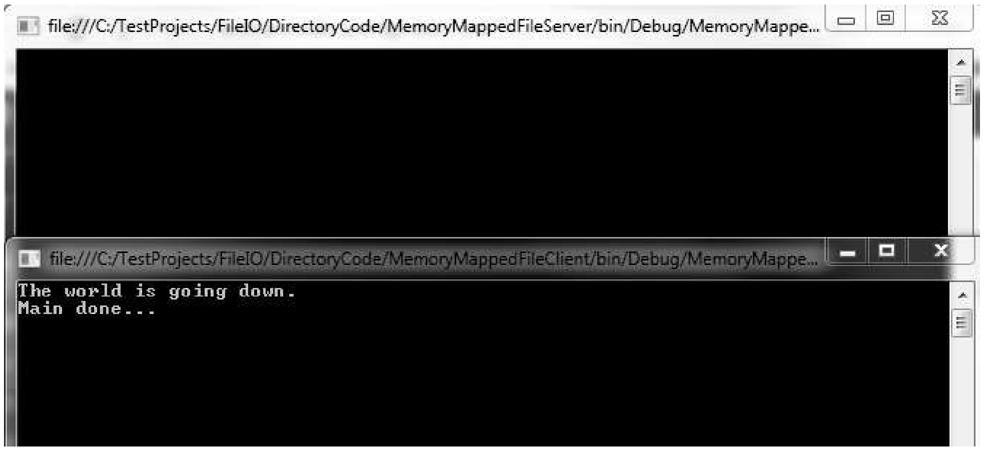
```
static void Main(string[] args)
{
    using (MemoryMappedFile memoryMappedFile = MemoryMappedFile.OpenExisting("news-channel"))
    {
        using (MemoryMappedViewAccessor viewAccessor = memoryMappedFile.CreateViewAccessor())
        {
            byte[] bytes = new byte[100];
            int res = viewAccessor.ReadArray(0, bytes, 0, bytes.Length);
            string text = Encoding.UTF8.GetString(bytes).Trim('\0');
            Console.WriteLine(text);
        }
    }

    Console.WriteLine("Main done...");
    Console.ReadKey();
}
```

Advertisements

REPORT THIS AD

Above we call the OpenExisting method to locate an existing memory-mapped file by its name. If it doesn’t exist the method throws an exception. You can put the above code into another C# console application and call it client. Run the server first and then the client before Thread.Sleep finishes in the server. The client should be able to read the message:


file:///C:/TestProjects/FileIO/DirectoryCode/MemoryMappedFileServer/bin/Debug/MemoryMappe...
file:///C:/TestProjects/FileIO/DirectoryCode/MemoryMappedFileClient/bin/Debug/MemoryMappe...

```
The world is going down.
Main done...
```

(https://dotnetcodr.files.wordpress.com/2015/03/memory-mapped-file-interprocess-communication.png)

View the list of posts on Messaging here (https://dotnetcodr.com/messaging/).

Advertisements

Save time & resources on your development using STM32Cube with Azure RTOS.
STMicroelectronics

REPORT THIS AD

FILED UNDER .NET, MESSAGING TAGGED WITH C#, MESSAGING

About Andras Nemes
I'm a .NET/Java developer living and working in Stockholm, Sweden.