Program

Addition:

```
private static string _loggerDirectory = @"C:\LogTest";
```

Different loggers may need different path , thus I give a chance to start the logger with the location of our choice from the app.

LogComponent

Logger.cs:
The logger on its own uses the ILogger interface that can be used for different loggers.
It instantiates a Document Service, a Thread Service and a blockingcollection of LogLines, which will be used into the threadservice. The logger passes its instance to the ThreadService with all its dependencies.
The methods Write, StopWithFlush, StopWithoutFlush work as intended
I have added one more Method to the interface name StartLogger that creates the running thread of the logger
I get Instances of the dependencies of the logger in case they need to be used. For testing proposes also.
I use the blockingCollections because they provide blocking and bounding capabilities for thread-safe collections. Plus the have autonomous try and catch exceptions while adding or taking from the collection.
The separation of the code between Document Service and Thread Service makes the code easier to read, more scalable, reusable and easier to be used on tests

DocumentService.cs:
The document service is responsible for conducting any action regarding the documents
Creates the directory, files and add lines with the help of the Document Helper. As above the reusable code has been separated from the service class and added into the helper. In the AddLogLine method I check the date in case of a different day than our instance. If it's true, I create a file and I update the instance with the help of the helper

DocumentHelper.cs:
The worker-code to create directories, files or add lines with empty try catch exceptions to ensure the continuation of the main app. In the future these exceptions can be enriched. A try to actually log the error in case of null LogLine has been successfully made on the DocumentService.AddLogLine. An Exceptions.cs file could be useful.

ThreadService.cs:
The Mainloop runs until the application ends or one of the designated signals is given. I have used the blockingCollections to create a producer-consumer pattern based the pending messages and write any message consumed into the file. I have created a ManualResetEvent to signal the thread in the case of `StopThreadServiceAndFlush. The join method works as intended for the case of StopThreadServiceAndWithoutFlush`

Future thoughts for the LogComponent

Create a ILogFactory to generalize the different loggers based on their type and maybe set different thread adapters based on their id if more than one loggers are run from the same app.
Rework the logger by using the log4net framework.

LogComponentTests

I lack of knowledge on how to test threads and I had the following problems:

1. In the logger occasions where I tried to test the write methods the unit test session remains pending without being able to debug it
2. When I tried to assert the files and check if the code was written correctly I had the issue that file was used by another process

All in all, I tried to create the specific test cases based on my thought even if they do not run correctly and write any comments needed. The frameworks that I have used and could be great if I knew better, are FluentAssertions and Moq.