Control File System Filter Driver Programming

Introduction

A file system filter driver is an optional driver that adds value to or modifies the behavior of a file system. A file system filter driver is a kernel-mode component that runs as part of the Microsoft Windows NT executive. A file system filter driver can filter I/O operations for one or more file systems or file system volumes. Depending on the nature of the driver, filter can mean log, observe, modify, or even prevent. Typical applications for file system filter drivers include antivirus utilities, encryption programs, and hierarchical storage management systems.

Developing file system filter driver is certainly a challenge job, it will take you months to learn and get used to the file system filter driver development. The EaseFilter File System Control Filter Driver SDK can simplify your development and to provide you with a robust and well-tested file system filter driver that works well with all versions and patch releases of the Windows operating systems supported by Microsoft, it provides you a fully tested framework of the filter driver to support all IRP, tracing, logging, communication with user mode application, even you are a user mode developer and only knows c#, c++ or any other language, you can fully control your file system without having the file system knowledge.

EaseFilter Control File System Filter Driver SDK

The EaseFilter control file system filter driver SDK includes two components (EaseFlt.sys and FilterAPI.dll), The EaseFlt.sys and FilterAPI.dll are different for 32bit and 64bit windows system. EaseFlt.sys is the file system filter driver which provides a complete, modular environment for building active file system filters. FilterAPI.dll is a user mode DLL which is responsible for the communication between filter driver and your user mode application, and it is also a wrapper DLL which exports the API to the user mode applications.

EaseFlt.sys includes the following modules:

Initialization module:
It is responsible for the registration of the filter driver, IRP.

Code tracing module:
WPP tracing for debug purpose.

Event log module:
It is responsible for the event log setting related features.

Context module:

It is responsible for the context tracking. It is very important module which tracks the file I/O operations.

Communication module:
It is responsible for the communication with the user mode applications.

Configuration module:
It is responsible for all the configuration settings.

Filter rules module:
It is responsible for the managed folders or files setup, which will be controlled by the control filter.

IRP base modules:
It is responsible for the IRP control code implementation if the IRP was registered in Initialization module.

FilterAPI DLL

It will create two communication channels with control filter driver, one is the data channel which provides the filter driver send data to the user mode application, another one is the control channel which is the user mode application send the control commands to the filter driver. This DLL provides the interfaces for the user mode applications to manipulate the control filter driver.

Use the control filter driver SDK step by step

Use EaseFilter SDK with C++ application

Copy the correct version (32bit or 64bit) EaseFlt.sys, FilterAPI.DLL, FilterAPI.h and FilterAPI.lib to your folder. FilterAPI.h file includes all the functions and structures used for connecting to the filter driver. WinDataStructures.h file is part of the structures of windows API which is used in the example, for more structures please reference Microsoft MSDN website.

For monitor filter, it will only display the file system call messages which include process Id, Thread Id, file name, user name, file system I/O type , etc.

For Control filter, the filter will block and wait for the response if that I/O was registered, so it is better handle this request as soon as possible, or it will block the system call.

Use EaseFilter SDK with C# application

Copy the correct version (32bit or 64bit) EaseFlt.sys , FilterAPI.DLL and ,*EaseFilter.cs* to your folder. EaseFilter.cs has the structures and APIs used for connecting to the filter driver.

Set up the filter

Install the filter driver with InstallDriver() method if the driver has not been installed yet. After filter driver was installed, the filter was loaded, if not you can load the filter with command “Fltmc load EaseFlt” in dos prompt. To remove the filter driver from the system, call UninstallDriver() method.

Start the filter

1. Activate the filter with API SetRegistrationKey(). You can buy a license key with the link: http://www.EaseFilter.com/Order.htm or email us info@EaseFilter.com to request a trial license key

2. After register the callback function with API RegisterMessageCallback, filter is started.

BOOL ret = RegisterMessageCallback(FilterConnectionThreadsCount, MessageCallback, DisconnectCallback);

3. Setup the filter configuration after filter was started. First select the filter type, then add filter rule and register the I/O request:

BOOL ret = SetFilterType(FILE_SYSTEM_CONTROL);

BOOL ret =
AddFilterRule((~ALLOW_OPEN_WITH_WRITE_ACCESS)&ALLOW_MAX_RIGHT_ACCESS ,L" C:\\MyMonitorFolder",L"");*

BOOL ret = RegisterIORequest(POST_CREATE|POST_CLEANUP);

We provide C++ example and C# example to demonstrate how to use the EaseFilter File System Monitor and Control Filter.

Control file system I/O is very easy with the EaseFilter Control File System Filter Driver SDK.Here is some examples to demostrate how to use it:

1. Not allow modification for .doc extension files in folder c:\protectFolder, you can do the following setting:

```
//Here is the example which you can't modify all the .doc files, can't write, can't change security, can't change file information ( creation time,last write time,file attributes..)
ULONG accessFlag = (~ALLOW_WRITE_ACCESS)&
(~ALLOW_SET_INFORMATION)& (~ALLOW_SET_SECURITY_ACCESS)&
ALLOW_MAX_RIGHT_ACCESS;
WCHAR* filterMask = L"c:\\protectFolder\\*.doc";

AddFilterRule(accessFlag,filterMask,L"");
```

2. Exclude all files in folder c:\protectFolder\excludeFolder from all your filter rules:

```
//Here is the example show you how to exclude the folder from the filter rule.
//The filter driver will by pass all the I/O for folder c:\protectFolder\excludeFolder

ULONG accessFlag = EXCLUDE_FILTER_RULE;
WCHAR* filterMask = L"c:\\protectFolder\\excludeFolder\\*";
```

```
AddFilterRule(accessFlag,filterMask,L"");
```

3. Reparse open for all the files in folder c:\test to the folder d:\reparse:

```
//Here is the reparse example, when you open the files in folder
c:\test, it will open the same file name in the folder d:\reparse

ULONG accessFlag = REPARSE_FILE_OPEN;
WCHAR* filterMask = L"c:\\test\\*";
WCHAR* reparseMask = L"d:\\reparse\\*";

AddFilterRule(accessFlag,filterMask, reparseMask);
```

4. Hide all files with extension .txt from folder c:\test, when you browse the folder, all the .txt files will be hidden from the file list.

```
ULONG accessFlag = HIDE_FILES_IN_DIRECTORY_BROWSING;
WCHAR* filterMask = L"c:\\test\\*";
WCHAR* reparseMask = L"*.txt";

AddFilterRule(accessFlag,filterMask,reparseMask);
```

5. Not allow file information modification for all files in folder c:\protectFolder:

```
//Here is the example you can't change the file information( file
time, attribute, file size..) for all the files in folder
c:\protectFolder
//remove set information access access permission
ULONG accessFlag = (~ALLOW_SET_INFORMATION) &
ALLOW_MAX_RIGHT_ACCESS;
WCHAR* filterMask = L"c:\\protectFolder\\*";

AddFilterRule(accessFlag,filterMask,L"");
```

6. Not allow renaming file operation for all files in folder c:\protectFolder:

```
ULONG accessFlag = (~ALLOW_FILE_RENAME) & ALLOW_MAX_RIGHT_ACCESS;
WCHAR* filterMask = L"c:\\protectFolder\\*";

AddFilterRule(accessFlag,filterMask,L"");
```

7. Not allow file deletion for all files in folder c:\protectFolder:

```
ULONG accessFlag = (~ALLOW_FILE_DELETE) & ALLOW_MAX_RIGHT_ACCESS;
WCHAR* filterMask = L"c:\\protectFolder\\*";

AddFilterRule(accessFlag,filterMask,L"");
```

8. Not allow changing the file size for all files in folder c:\protectFolder:

```
ULONG accessFlag = (~ALLOW_FILE_SIZE_CHANGE) &
ALLOW_MAX_RIGHT_ACCESS;
WCHAR* filterMask = L"c:\\protectFolder\\*";

AddFilterRule(accessFlag,filterMask,L"");
```

9. Not allow browsing the folder c:\protectFolder, when browse the folder, you will get access denied error.

```
ULONG accessFlag = (~ALLOW_DIRECTORY_LIST_ACCESS) &
ALLOW_MAX_RIGHT_ACCESS;
WCHAR* filterMask = L"c:\\protectFolder\\*";

AddFilterRule(accessFlag,filterMask,L"");
```

Here is the examples how to modify the I/O data before it passes to the file system or after returns from the file system.

1. Test read request data modification.

Here is the example when you open and read the file's content from the folder c:\protectFolder, it will invoke your call back function, And you can return your own customized data.

Set the filter rule and register the pre-read requests:

```
ULONG accessFlag = ALLOW_OPEN_WITH_READ_ACCESS
|ALLOW_READ_ACCESS;
WCHAR* filterMask = L"c:\\protectFolder\\*";
AddFilterRule(accessFlag,filterMask,L"");

ULONG requestRegistration =
PRE_FASTIO_READ|PRE_CACHE_READ|PRE_NOCACHE_READ|PRE_PAGING_IO_READ|
POST_FASTIO_READ| POST_CACHE_READ| POST_NOCACHE_READ|
POST_PAGING_IO_READ;
RegisterIoRequest(requestRegistration);
```

2. Test write request data modification. Change the write data before it goes down to the file system.

The AddFilterRule is the same as read request test.

```
ULONG requestRegistration = PRE_FASTIO_WRITE | PRE_CACHE_WRITE |
PRE_NOCACHE_WRITE | PRE_PAGING_IO_WRITE;
RegisterIoRequest(requestRegistration);
```

3. Test query information request.

The AddFilterRule is the same as read request test.

```
ULONG requestRegistration = PRE_QUERY_INFORMATION | POST_QUERY_INFORMATION;
```

```
RegisterIoRequest(requestRegistration);
```

4. Test set information request.

The AddFilterRule is the same as read request test.

```
ULONG requestRegistration = PRE_SET_INFORMATION;
RegisterIoRequest(requestRegistration);
```

5. Test change the directory browse request data.

The AddFilterRule is the same as read request test.

```
ULONG requestRegistration = POST_DIRECTORY;
RegisterIoRequest(requestRegistration);
```