

Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого

—
Институт прикладной механики и математики
Кафедра «Информационная безопасность компьютерных систем»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2
ИССЛЕДОВАНИЕ СПОСОБОВ ХРАНЕНИЯ ФАЙЛОВ И КЛЮЧЕЙ
РЕЕСТРА В ПРОГРАММНОМ ОБЕСПЕЧЕНИИ
по дисциплине «Теория обнаружения вторжений»

Выполнили
студент гр. 53609/3
студент гр. 53609/3

М.А. Латышев
П.О. Семёнов

Преподаватель

А.И. Печенкин

Санкт-Петербург
2018

1 ФОРМУЛИРОВКА ЗАДАНИЯ

Реализовать драйвер-фильтр, скрывающий указанный объект файловой системы ОС Windows.

Вариант №4.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Рассмотрим схему обращению приложения к устройству. В данном случае устройство – жесткий диск.

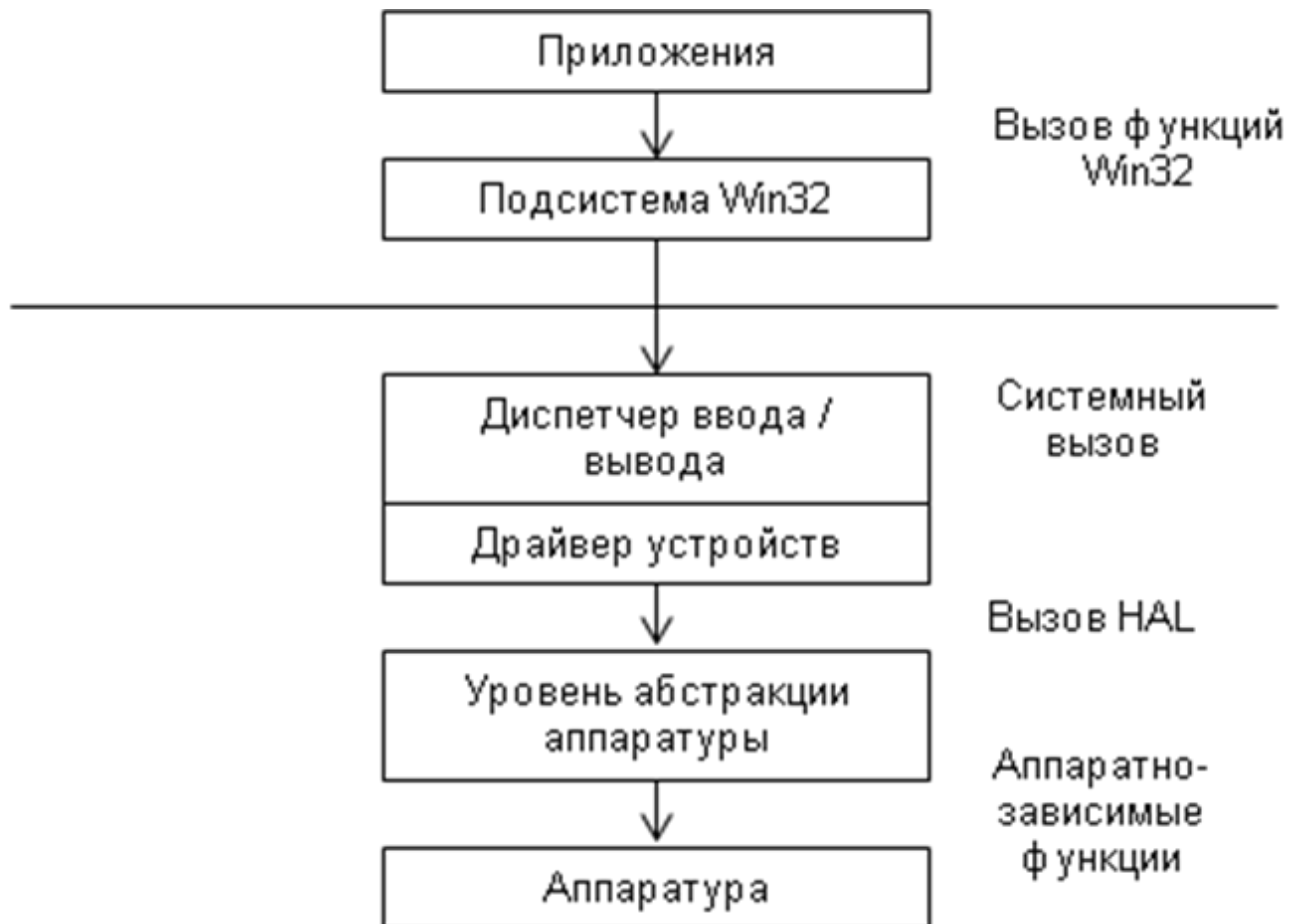


Рисунок 1 – схема обращения

Для обработки обращений воспользуемся драйвер-фильтром. Драйвер-фильтр, как и любой драйвер работает в режиме ядра.

Теперь покажем функционирование и расположение фильтра.

Пользовательские запросы ввода/вывода

User Mode

Kernel Mode

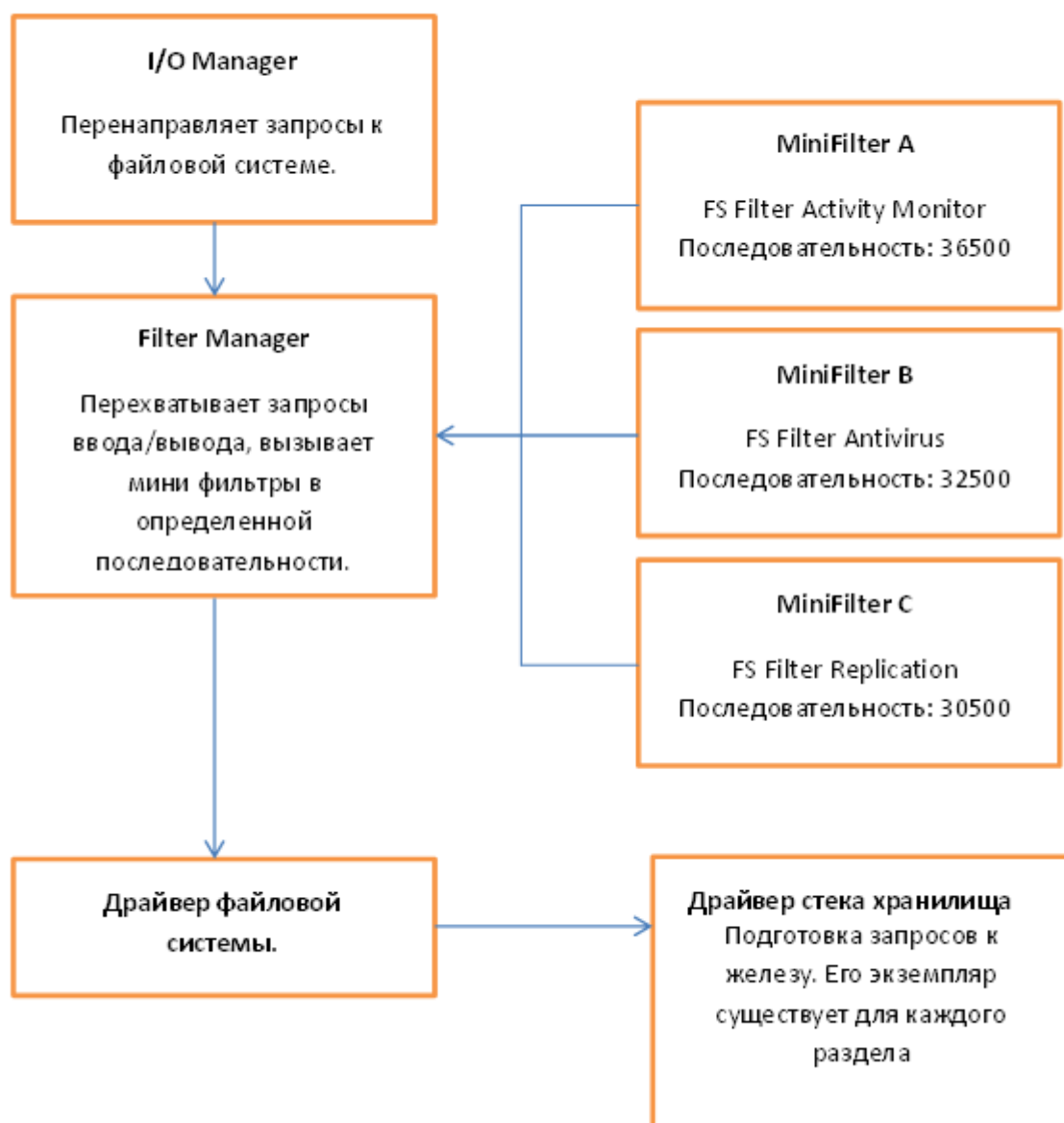


Рисунок 2 – схема функционирования драйвер-фильтра

Таким образом, любой запрос от пользовательского приложения далее обрабатывается в пространстве ядра соответствующим драйвером.

Реализовав драйвер-фильтр файловой системы получим возможность контролировать исполняемые операции в файловой системе, в частности, контролировать перечисление файлов и других объектов в директории.

3 РЕЗУЛЬТАТЫ РАБОТЫ

Для разработки драйвер-фильтра была использована ОС Windows 10 x64, Visual Studio 2017 с установленным Driver Development Kit, который позволяет создавать драйверы для операционной системы.

Для разработки драйверов требуется создать специальный проект, при генерации создает не только файл драйвера, но и INF-файл установки.

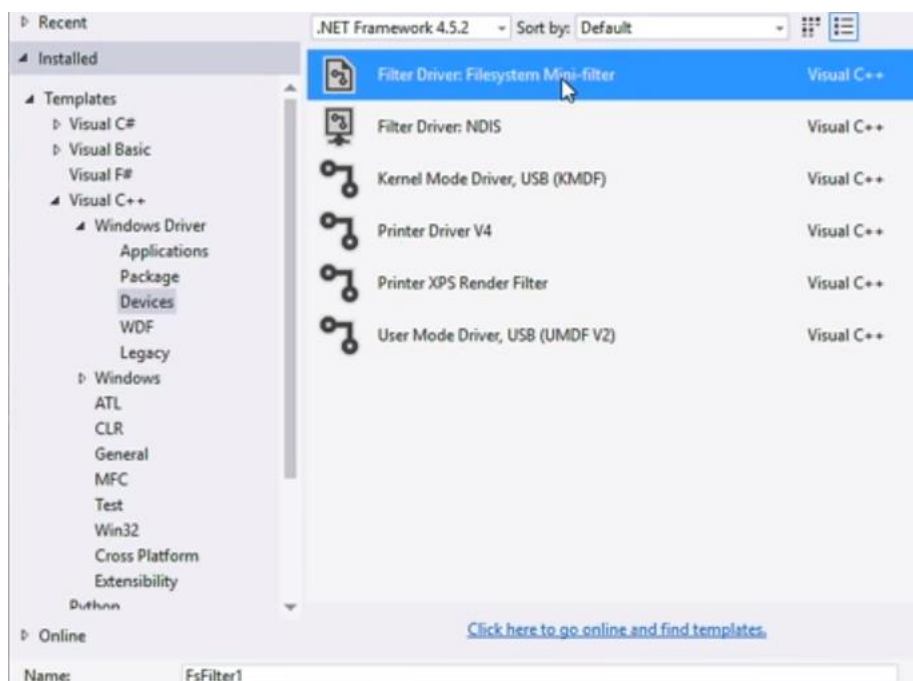


Рисунок 3 – проект драйвера

Для разработки драйвера требуется подключить специальные библиотеки “fltKernel.h”.

Минимальный драйвер-фильтр должен быть зарегистрирован в системе при помощи функции `FltRegisterFilter`, которая принимает на вход необходимые для функционирования параметры.

По окончании работы драйвер должен быть «выгружен» при помощи `FltUnregisterFilter`.

При регистрации фильтра необходима инициализация структуры `FLT_REGISTRATION`, главный компонент которой – массив callback-функций, которые выполняются в соответствии с событиями:

```
FLT_OPERATION_REGISTRATION Callbacks[] =  
{  
    { IRP_MJ_CREATE, 0, MiniPreCreate, MiniPostCreate },  
    { IRP_MJ_WRITE, 0, MiniPreWrite, NULL },
```

```
{ IRP_MJ_OPERATION_END }  
};
```

Так, обязательно должны быть события IRP_MJ_CREATE, IRP_MJ_OPERATION_END, где первое отвечает за функции, исполняемые при запуске драйвера. Последние два параметра каждой структуры отвечают за функции, исполняемые до (pre) и после (post) исполнения указанного события. Так, для события записи (IRP_MJ_WRITE) определена функция только до исполнения записи – MiniPreWrite.

Запуск фильтра производится функцией FltStartFiltering.

Для контроля над содержимым директорий нас интересует функция ZwQueryDirectoryFile, которая связана с событием IRP_MJ_DIRECTORY_CONTROL. Функция, фильтрующая указанное событие должна быть типа post – поскольку в pre-функции нет информации о файлах, так как запрос еще не был действительно исполнен.

Каждая callback-функция принимает указатель (PFLT_CALLBACK_DATA Data) на требуемые данные. Для работы с содержимым директорий нас интересует DirectoryBuffer, который и хранит всю информацию о директории. Стоит заметить, что информация доступна в нескольких представлениях, каждое из которых требуется обработать.

Получая сведения о директории, следует пройти по каждой записи буфера, и по имени определить файл, который требуется скрыть:

```
if (wcsstr(fileName.Buffer, HideFileName.Buffer) != NULL)
```

Если есть вхождение, следует убрать его из буфера, корректно перезаписать буфер и вернуть FLT_POSTOP_FINISHED_PROCESSING.

Так, в буфере больше не будет содержаться сведений о файле, который требовалось скрыть.

Для установки драйвера требуется изменить INF-файл, после чего с его же помощью установить драйвер.

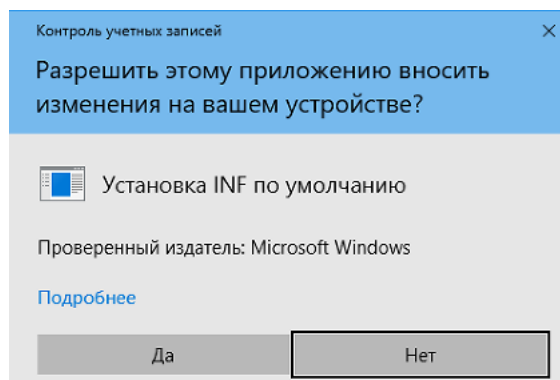


Рисунок 4 – установка драйвера из INF-файла

Теперь драйвер установлен в систему, запустить его можно из командной строки с административными привилегиями:

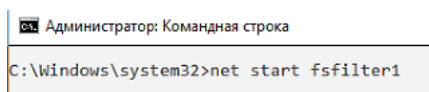


Рисунок 5 – запуск драйвера

Поскольку драйвер работает в режиме ядра, сведения о его работе будем получать посредством DebugView:

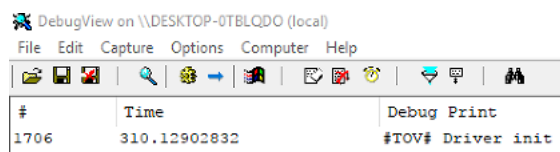


Рисунок 6 – сообщения DebugView

Для начала зададим драйверу начальный файл для скрятия – “THISFILE.TXT”, попытаемся найти его в проводнике, при этом наблюдая за выводом в DebugView:

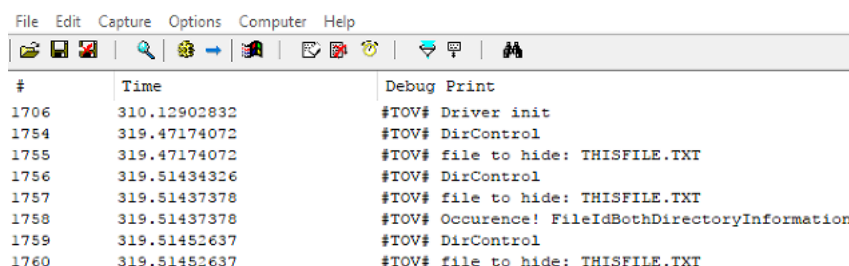


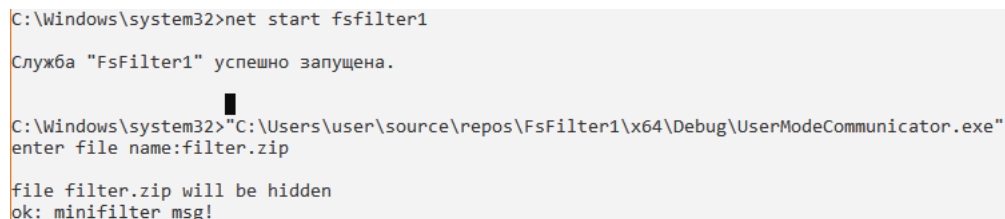
Рисунок 7 – вывод DebugView при скрятии файла

Из вывода можно узнать, что файл был успешно опознан, а структура при перечислении файлов (там, где сработал обнаружение) – FileIdBothDirectoryInformation. Таким образом можно скрыть любой файл в операционной системе.

Для удобства и во избежание перестроения проекта было бы удобно установить связь между драйвером, функционирующем в режиме ядра, и приложением, в которое пользователь может ввести имя нового файла.

Для этого создадим простейшее консольное приложение, использующее "fltUser.h". Приложение установит канал связи с драйвером и передаст ему сообщение. За установку канал отвечает функция FilterConnectCommunicationPort, а за отправку сообщения – FilterSendMessage. Также требуется внести соответствующие изменения в код самого драйвера – установить порт, установить callback-функции и обработку сообщения.

Введем в приложение новое имя для скрывтия:



```
C:\Windows\system32>net start fsfilter1

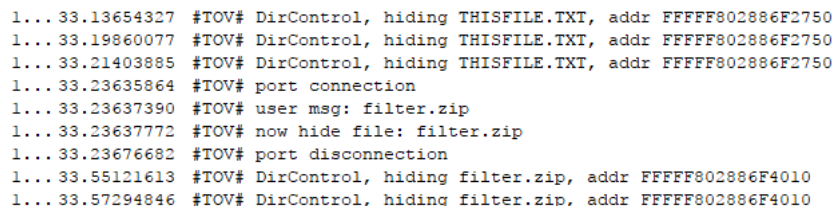
Служба "FsFilter1" успешно запущена.

C:\Windows\system32>"C:\Users\user\source\repos\FsFilter1\x64\Debug\UserModeCommunicator.exe"
enter file name:filter.zip

file filter.zip will be hidden
ok: minifilter msg!
```

Рисунок 8 – пользовательское приложение

Драйвер в ответ всегда отвечает сообщением «minifilter msg!». Теперь обратимся к выводу DebugView:



```
1...33.13654327 #TOV# DirControl, hiding THISFILE.TXT, addr FFFFF802886F2750
1...33.19860077 #TOV# DirControl, hiding THISFILE.TXT, addr FFFFF802886F2750
1...33.21403885 #TOV# DirControl, hiding THISFILE.TXT, addr FFFFF802886F2750
1...33.23635864 #TOV# port connection
1...33.23637390 #TOV# user msg: filter.zip
1...33.23637772 #TOV# now hide file: filter.zip
1...33.23676682 #TOV# port disconnection
1...33.55121613 #TOV# DirControl, hiding filter.zip, addr FFFFF802886F4010
1...33.57294846 #TOV# DirControl, hiding filter.zip, addr FFFFF802886F4010
```

Рисунок 9 – вывод DebugView

При установлении канала связи, в выводе наблюдаются сообщения об открытии и закрытии порта, а также выводится сообщения от пользователя, которое незамедлительно становится именем файла, который требуется скрыть.

Предыдущий скрываемый файл снова появится, а новый будет скрыт. Остановить драйвер и снова показать все файлы можно при помощи команды net stop fsfilter1.

4 ВЫВОДЫ

Изучены механизмы хранения и обработки файлов в операционной системе Windows, структура вызовов и компонентов режима ядра, отвечающих за работу с файловой системой.

Разработан драйвер-фильтр, фильтрующий запросы пользовательских приложений и скрывающий указанный файл при помощи изменений в буфере, полученном от нижележащих функций (процедур драйверов).

Разработана пользовательская программа, осуществляющая коммуникацию с драйвером, передавая и получая информацию.

Данный механизм может быть использован с целью скрыть вредоносное ПО, что характерно для руткитов. Но для установки драйверов требуются административные привилегии, а в более поздних версиях Windows, драйвера подписываются и проходят проверку на подлинность.

ПРИЛОЖЕНИЕ

Клиентская программа

```
#include <Windows.h>
#include <fltUser.h>
#include <fltUserStructures.h>

#include <fltdefs.h>
#include <stdio.h>

#pragma comment(lib, "user32.lib")
#pragma comment(lib, "kernel32.lib")
#pragma comment(lib, "fltlib.lib")
#pragma comment(lib, "fltmgr.lib")

HANDLE port = NULL;

void main()
{
    DWORD byterec = 0;
    WCHAR buffer[255] = { 0 };
    char recbuffer[255] = { 0 };

    printf("enter file name:");
    scanf("%ws", &buffer);
    printf("\nfile %ws will be hidden\n", buffer);

    FilterConnectCommunicationPort(L"\\mf", 0, NULL, 0, NULL, &port);

    if (FilterSendMessage(port, buffer, wcslen(buffer), recbuffer, 255, &byterec))
    {
        printf("not ok");
    }
    else
    {
        printf("ok: %s \n", recbuffer);
    }

    CloseHandle(port);
}
```

Драйвер

```
#include <fltKernel.h>
#include <dontuse.h>
#include <suppress.h>

PFLT_PORT port;
PFLT_PORT ClientPort;
UNICODE_STRING HideFileName;
WCHAR FN[255] = { 0 };
PFLT_FILTER FilterHandle = NULL;
NTSTATUS MiniUnload(FLT_FILTER_UNLOAD_FLAGS Flags);
FLT_POSTOP_CALLBACK_STATUS MiniPostCreate(PFLT_CALLBACK_DATA Data, PCFLT_RELATED_OBJECTS
FltObjects, PVOID* CompletionContext, FLT_POST_OPERATION_FLAGS Flags);
FLT_PREOP_CALLBACK_STATUS MiniPreCreate(PFLT_CALLBACK_DATA Data, PCFLT_RELATED_OBJECTS
FltObjects, PVOID* CompletionContext);
FLT_PREOP_CALLBACK_STATUS MiniPreWrite(PFLT_CALLBACK_DATA Data, PCFLT_RELATED_OBJECTS
FltObjects, PVOID* CompletionContext);
```

```

FLT_POSTOP_CALLBACK_STATUS MiniPostDirControl(PFLT_CALLBACK_DATA Data,
PCFLT_RELATED_OBJECTS FltObjects, PVOID* CompletionContext, FLT_POST_OPERATION_FLAGS
Flags);

const FLT_OPERATION_REGISTRATION Callbacks[] =
{
    { IRP_MJ_CREATE,0,MiniPreCreate,MiniPostCreate },
    //{ IRP_MJ_WRITE,0,MiniPreWrite,NULL },
{ IRP_MJ_DIRECTORY_CONTROL,0,NULL,MiniPostDirControl },
{ IRP_MJ_OPERATION_END }
};

const FLT_REGISTRATION FilterRegistration =
{
    sizeof(FLT_REGISTRATION),
    FLT_REGISTRATION_VERSION,
    0,
    NULL,
    Callbacks,
    MiniUnload,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL
};

NTSTATUS MiniConnect(PFLT_PORT clientport, PVOID serverportcookie, PVOID context, ULONG
size, PVOID connectioncookie)
{
    ClientPort = clientport;
    KdPrint(("#TOV# port connection\n"));
    return STATUS_SUCCESS;
}

VOID MiniDisconnect(PVOID connectioncookie)
{
    KdPrint(("#TOV# port disconnection\n"));
    FltCloseClientPort(FilterHandle, &ClientPort);
}

NTSTATUS MiniUnload(FLT_FILTER_UNLOAD_FLAGS Flags)
{
    KdPrint(("#TOV# Driver Unload \n"));
    FltCloseCommunicationPort(port);
    FltUnregisterFilter(FilterHandle);

    return STATUS_SUCCESS;
}

NTSTATUS MiniSendRec(PVOID portcookie, PVOID InputBuffer, ULONG InputBufferLength, PVOID
OutputBuffer, ULONG OutputBufferLength, PULONG RetLength)
{
    PCHAR msg = "minifilter msg!";
    KdPrint(("#TOV# user msg: %ws \n", (PCHAR)InputBuffer));

    strcpy((PCHAR)OutputBuffer, msg);

    wcsncpy(FN, InputBuffer);

```

```

    RtlInitUnicodeString(&HideFileName, FN);

    KdPrint(("#TOV# now hide file: %ws \n", HideFileName.Buffer));

    return STATUS_SUCCESS;
}
FLT_POSTOP_CALLBACK_STATUS MiniPostCreate(PFLT_CALLBACK_DATA Data, PCFLT_RELATED_OBJECTS
FltObjects, PVOID* CompletionContext, FLT_POST_OPERATION_FLAGS Flags)
{
    //KdPrint(("#TOV# post-create runs \n"));
    return FLT_POSTOP_FINISHED_PROCESSING;
}

FLT_PREOP_CALLBACK_STATUS MiniPreCreate(PFLT_CALLBACK_DATA Data, PCFLT_RELATED_OBJECTS
FltObjects, PVOID* CompletionContext)
{
    PFLT_FILE_NAME_INFORMATION FileNameInfo;
    NTSTATUS status;
    WCHAR Name[256] = { 0 };

    status = FltGetFileNameInformation(Data, FLT_FILE_NAME_NORMALIZED |
FLT_FILE_NAME_QUERY_DEFAULT, &FileNameInfo);

    if (NT_SUCCESS(status))
    {
        status = FltParseFileNameInformation(FileNameInfo);

        if (NT_SUCCESS(status))
        {
            if (FileNameInfo->Name.MaximumLength < 255)
            {
                //RtlCopyMemory(Name, FileNameInfo->Name.Buffer, FileNameInfo-
>Name.MaximumLength);
                //KdPrint(("#TOV# create file %ws \n", Name));
            }

            FltReleaseFileNameInformation(FileNameInfo);
        }
        return FLT_PREOP_SUCCESS_WITH_CALLBACK;
    }
}

```