

## Jak system Windows tworzy pliki?

Celem tego sprawozdania jest zaprezentowanie elementów systemu I/O biorących udział w tworzeniu pliku. Ułatwimy proces badania tego zjawiska poprzez dodanie do niego jednego filtra, który będzie widoczny jak instancja sterownika *FltMgr*.

Aby prześledzić ten proces wyobraźmy sobie, że program użytkownika wykonuje wywołanie w postaci:

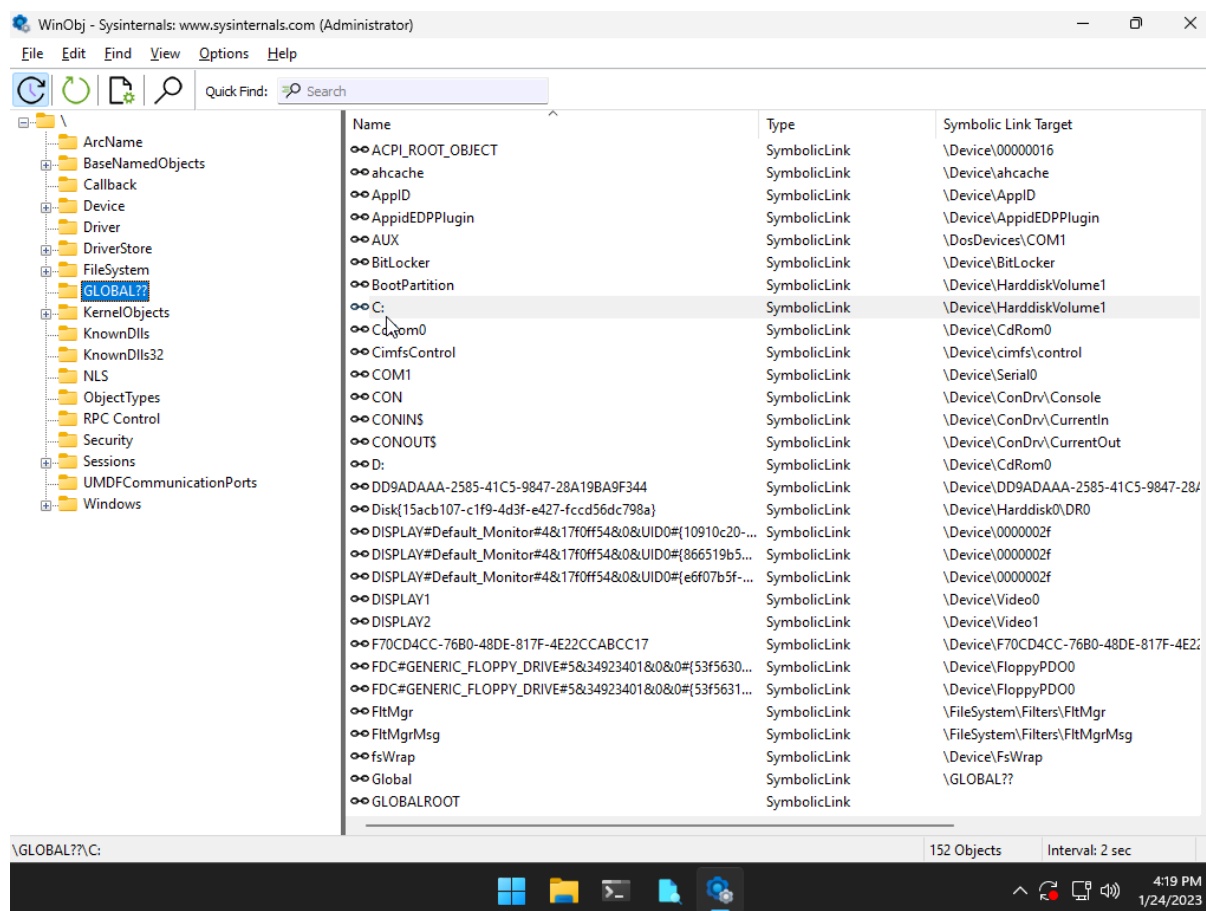
```
hFile = CreateFile("C:\\plik.txt",           // name of the write
                  GENERIC_WRITE,           // open for writing
                  0,                       // do not share
                  NULL,                   // default security
                  CREATE_NEW,             // create new file only
                  FILE_ATTRIBUTE_NORMAL,  // normal file
                  NULL);                 // no attr. template
```

Warto zwrócić w tym miejscu uwagę, że funkcja *CreateFile* nie otwiera tylko plików w kontekście systemu plików, ale też *wirtualne pliki*, które reprezentują różnego rodzaju urządzenia fizyczne bądź logiczne. Przykładem takiego, będzie przedstawione później urządzenie *Ntfs*.

## Podejście początkowe

Mając podstawową wiedzę na temat funkcji I/O *managera* systemu *Windows* możemy się spodziewać, że wywołanie *CreateFile* spowoduje stworzenie odpowiedniego *IRP* (*Interrupt Request Packet*), które zostanie przekazane do urządzenia obsługującego ścieżkę *C:\plik.txt*.

Korzystając z narzędzia *WinObj* z pakietu *sysinternals* możemy sprawdzić w jaki sposób jest rozwijana ścieżka zaczynająca się od *C:\* korzystając z tablicy *GLOBAL??*.



Możemy zobaczyć, że C:\ jest rozwijane do \\Device\\HarddiskVolume1. Teraz możemy sprawdzić jakie sterowniki będą obsługiwać to przerwanie. Zakładając, że jest to model warstwowy wykorzystamy komendę !devstack w WinDbg, która wyświetla stos tych sterowników.

```
0: kd> !devstack HarddiskVolume1
```

!DevObj	!DrvObj	!DevExt	ObjectName
ffffdb0586707030	\\Driver\\volsnap	ffffdb0586707180	
ffffdb05865c3d20	\\Driver\\volume	ffffdb05865c3e70	
ffffdb05866078d0	\\Driver\\rdyboost	ffffdb0586607a20	
ffffdb05866098d0	\\Driver\\iorate	ffffdb0586609a20	
ffffdb0586705030	\\Driver\\fvevol	ffffdb0586705180	
> fffffdb058660e8f0	\\Driver\\volmgr	ffffdb058660ea40	HarddiskVolume1

Ta obserwacja może być dla nas zaskoczeniem - na powyższym stosie nie ma żadnego sterownika związanego z systemem plików czy fizycznym urządzeniem dyskowym. Jest to spowodowane tym, że przypisanie woluminowi litery - w tym przypadku C - nie wiąże się z powiązaniem go z systemem plików.

## Dalsza obsługa przerwania IRP\_MJ\_CREATE

Aby dowiedzieć się, jaki system plików obsługuje ten wolumin, musimy skorzystać z widocznego powyżej sterownika *volmgr*, który zbada strukturę nazywaną *volume parameter block* (VPB).

Wszystkie *IRP* zawierające ścieżki do plików na danym urządzeniu skierowane do *volmgr* są przekierowywane (poprzez ponowne wykorzystanie *IRP* lub stworzenie nowego) do odpowiedniego sterownika systemu plików.

Możemy przyjrzeć się temu procesowi za pomocą komendy *!drvobj volmgr*, która służy do badania sterowników i przedstawia nam dostępne informacje na ich temat.

```
0: kd> !drvobj volmgr
Driver object (ffff960c40734e10) is for:
  \Driver\volmgr
```

Driver Extension List: (id , addr)

```
Device Object list:
ffff960c40860870  fffff960c4073aba0
```

Upraszczając, pierwszy *device object* w powyższej liście reprezentuje nasz wolumin C, a drugi to urządzenie kontrolne *volmgr'a*. Następnym krokiem będzie zbadanie tego obiektu:

```
0: kd> !devobj fffff960c40860870
Device object (ffff960c40860870) is for:
  HarddiskVolume1 \Driver\volmgr DriverObject fffff960c40734e10
Current Irp 00000000 RefCount 40417 Type 00000007 Flags 00201150
Vpb 0xffff960c408d5ab0 SecurityDescriptor fffffba0bda859060 DevExt fffff960c408609
ExtensionFlags (0x00000800)  DOE_DEFAULT_SD_PRESENT
Characteristics (0x00020000)  FILE_DEVICE_ALLOW_APPCONTAINER_TRAVERSAL
AttachedDevice (Upper) fffff960c408ef030 \Driver\fvevol
Device queue is not busy.
```

W wyniku wykonania tej operacji, poznaliśmy uchwyt do struktury *VPB* naszego woluminu, który możemy zbadać poleceniem *!vpb*:

```
0: kd> !vpb 0xffff960c408d5ab0
Vpb at          0xffff960c408d5ab0
```

```
Flags:          0x1 mounted
DeviceObject: 0xffff960c4099f030 (dt nt!DEVICE_OBJECT)
RealDevice:   0xffff960c40860870 (dt nt!DEVICE_OBJECT)
RefCount:     40417
Volume Label: ""
```

To polecenie zwraca nam między innymi uchwyt na *DeviceObject*, który reprezentuje urządzenie do którego dalej zostanie przekazane wysłane przez nas *IRP*. Możemy je zbadać poleceniem *!devstack*:

```
0: kd> !devstack fffff960c4099f030
!DevObj          !DrvObj          !DevExt          ObjectName
fffff960c40847d20 \FileSystem\FltMgr fffff960c40847e70
> fffff960c4099f030 \FileSystem\Ntfs fffff960c4099f1b0
```

Udało się nam to, czego oczekiwaliśmy na samym początku - odwołanie do sterownika *Ntfs* oraz wspomnianego na początku górnego filtra *FltMgr*. Kolejnym etapem przetwarzania tego żądania będzie wysłanie nowego *IRP* do sterownika *volmgr* przez sterownik *Ntfs*, ale tym razem korzystającego z interfejsu blokowego tego urządzenia zamiast dostępu do danego pliku jak miało to miejsce wcześniej.

**Prawdopodobnie** będzie to miało miejsce poprzez wywołanie *IoCreateFile*, *IoCreateFileSpecifyDeviceObjectHint*, *ZwCreateFile*, lub *ZwOpenFile*, którego celem będzie teraz *\Device\HarddiskVolume1* - w przeciwieństwie do *\Device\HarddiskVolume1\plik.txt*, które *volmgr* otrzymał na samym początku oraz różnego rodzaju operacje odczytu i zapisu na poziomie blokowym.

Ostatnim etapem będzie wysłanie przerwania do sterownika dysku. Możemy sprawdzić, jak będzie wyglądała jego obsługa. Wiemy, że urządzenie reprezentujące dysk, na którym jest wolumin C to *\Device\Harddisk0\DR0*. Na tej podstawie:

```
0: kd> !devstack Harddisk0\DR0
!DevObj          !DrvObj          !DevExt          ObjectName
fffffdb058661d8d0 \\Driver\partmgr fffffdb058661da20
> fffffdb0586663060 \\Driver\disk fffffdb05866631b0 DR0
fffffdb0586531050 \\Driver\storahci fffffdb05865311a0 00000029
!DevNode fffffdb0586536af0 :
DeviceInst is "SCSI\Disk&Ven_VBOX&Prod_HARDDISK\4&2617aeae&0&000000"
ServiceName is "disk"
```

## Podsumowanie

Na podstawie naszego małego “*dochodzenia*” możemy dojść do wniosku, że proces tworzenia pliku w systemie *Windows* wygląda następująco:

1. Wywołanie *CreateFile* korzystające ze ścieżki *C:\plik.txt*
2. Stworzenie *IRP* typu *IRP\_MJ\_CREATE* do urządzenia *\Device\HarddiskVolume1*
3. Obsługa *IRP* przez sterownik *volmgr*
4. Wysłanie *IRP* do urządzenia *Ntfs*
5. Stworzenie *IRP* opisującego operacje blokowe potrzebne do utworzenia pliku w systemie plików *Ntfs*. I wysłanie go do urządzenia *\Device\HarddiskVolume1*
6. *volmgr* wysyła żądanie zapisu na fizycznym dysku *\Device\Harddisk0\DR0*.

## Źródła

- Windows Internals, Part 1,2 (Developer Reference) - Russinovich, Mark; Allievi, Andrea; Ionescu, Alex; Solomon, David

## Kontakt

- <https://github.com/blazej-smorawski/windows-package-manager-report>