

Hibakeresés Windowson

Operációs rendszerek gyakorlati útmutató

Készítette: Micskei Zoltán

Utolsó módosítás: 2013.02.19.

A gyakorlat célja, hogy megismerjünk néhány alapvető eszközt, amivel különböző hibák okát lehet felderíteni, legyen az hibás szoftver, teljesítményprobléma vagy akár a számítógép összeomlása.

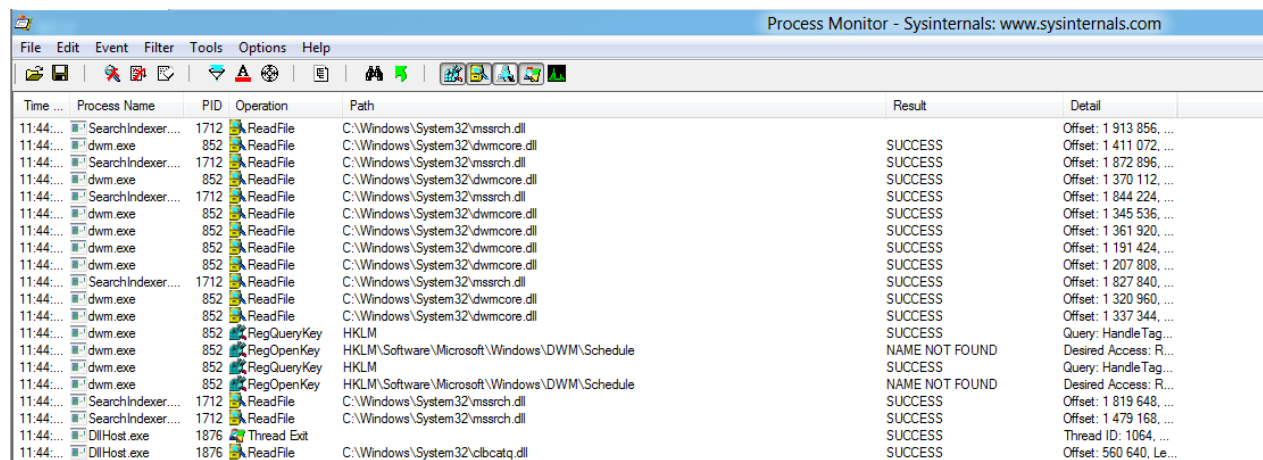
A feladatokat érdemes virtuális gépben végrehajtani, a tantárgy „Windows alapok” gyakorlata tartalmaz egy útmutatót, hogy hogyan hozunk létre egy megfelelő virtuális gépet.

1 Hozzáférési hibák vizsgálata

Ha arra gyanakszunk, hogy az adott hibát valami fájl hozzáférési probléma okozza (pl. nincs megfelelő jogosultsága az alkalmazásnak, nem talál valami keresett fájlt), akkor ilyen helyzetekben a Sysinternals csomag *Process Monitor* eszközét érdemes használni.

1.1 A Process Monitor bemutatása

A Process Monitor (1. ábra) rögzíti a rendszer összes fájl, registry és hálózati műveleteit, amit utána könnyedén szűrhetünk és elemezhetünk.



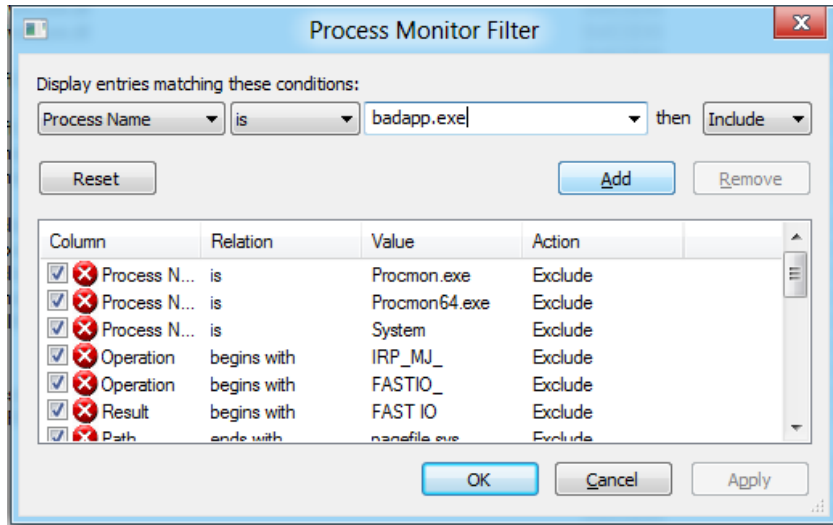
The screenshot shows the Process Monitor interface with a table of events. The table has columns for Time, Process Name, PID, Operation, Path, Result, and Detail. The events listed include file reads and registry queries performed by SearchIndexer and dwm.exe.

Time ...	Process Name	PID	Operation	Path	Result	Detail
11:44:...	SearchIndexer...	1712	ReadFile	C:\Windows\System32\vmssrch.dll		
11:44:...	dwm.exe	852	ReadFile	C:\Windows\System32\dwmcore.dll	SUCCESS	Offset: 1 913 856, ...
11:44:...	SearchIndexer...	1712	ReadFile	C:\Windows\System32\vmssrch.dll	SUCCESS	Offset: 1 411 072, ...
11:44:...	dwm.exe	852	ReadFile	C:\Windows\System32\dwmcore.dll	SUCCESS	Offset: 1 872 896, ...
11:44:...	SearchIndexer...	1712	ReadFile	C:\Windows\System32\vmssrch.dll	SUCCESS	Offset: 1 370 112, ...
11:44:...	dwm.exe	852	ReadFile	C:\Windows\System32\dwmcore.dll	SUCCESS	Offset: 1 844 224, ...
11:44:...	dwm.exe	852	ReadFile	C:\Windows\System32\dwmcore.dll	SUCCESS	Offset: 1 345 536, ...
11:44:...	dwm.exe	852	ReadFile	C:\Windows\System32\dwmcore.dll	SUCCESS	Offset: 1 361 920, ...
11:44:...	dwm.exe	852	ReadFile	C:\Windows\System32\dwmcore.dll	SUCCESS	Offset: 1 191 424, ...
11:44:...	dwm.exe	852	ReadFile	C:\Windows\System32\dwmcore.dll	SUCCESS	Offset: 1 207 808, ...
11:44:...	SearchIndexer...	1712	ReadFile	C:\Windows\System32\vmssrch.dll	SUCCESS	Offset: 1 827 840, ...
11:44:...	dwm.exe	852	ReadFile	C:\Windows\System32\dwmcore.dll	SUCCESS	Offset: 1 320 960, ...
11:44:...	dwm.exe	852	ReadFile	C:\Windows\System32\dwmcore.dll	SUCCESS	Offset: 1 337 344, ...
11:44:...	dwm.exe	852	RegQueryValue	HKLM	SUCCESS	Query: HandleTag...
11:44:...	dwm.exe	852	RegOpenKey	HKLM\Software\Microsoft\Windows\DWMSchedule	NAME NOT FOUND	Desired Access: R...
11:44:...	dwm.exe	852	RegQueryValue	HKLM	SUCCESS	Query: HandleTag...
11:44:...	dwm.exe	852	RegOpenKey	HKLM\Software\Microsoft\Windows\DWMSchedule	NAME NOT FOUND	Desired Access: R...
11:44:...	SearchIndexer...	1712	ReadFile	C:\Windows\System32\vmssrch.dll	SUCCESS	Offset: 1 819 648, ...
11:44:...	SearchIndexer...	1712	ReadFile	C:\Windows\System32\vmssrch.dll	SUCCESS	Offset: 1 479 168, ...
11:44:...	DllHost.exe	1876	Thread Exit		SUCCESS	Thread ID: 1064, ...
11:44:...	DllHost.exe	1876	ReadFile	C:\Windows\System32\clbcatq.dll	SUCCESS	Offset: 560 640, Le...

1. ábra: A Process Monitor felülete

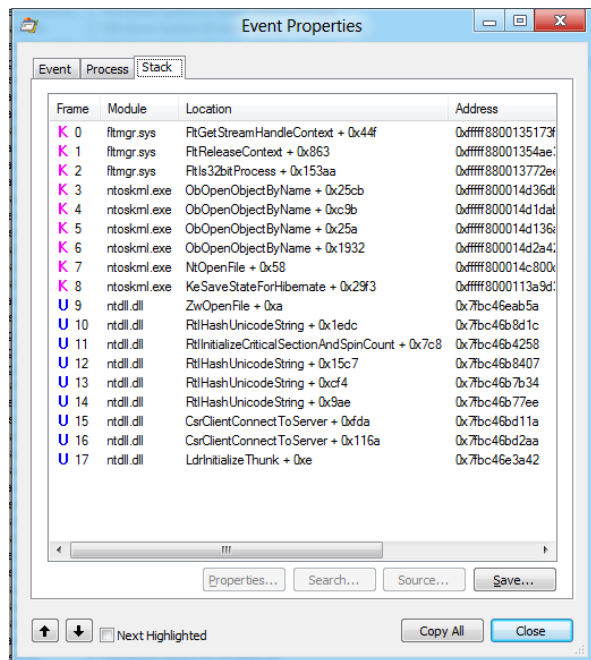
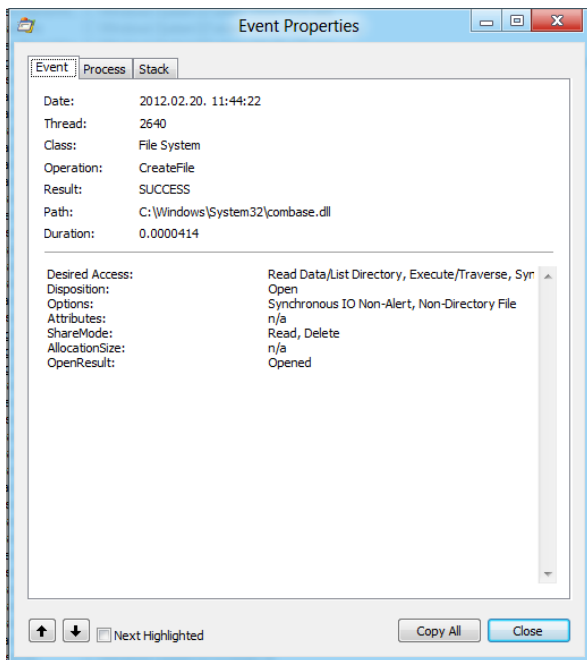
Alapesetben egy Windowson, ha látszólag semmi se fut, akkor is tipikusan több ezer esemény történik a háttérben. Ezért érdemes mindig valami megfelelő szűrőfeltételt beállítani (2. ábra). Lehet szűrni a program nevére, a folyamat azonosítójára vagy a művelet

visszatérési értékére (pl. NOT FOUND vagy ACCESS DENIED). Lehet részleges egyezésre is szűrni.



2. ábra: Szűrőfeltétel beállítása Process Monitorban

Egy-egy eseményről részletes adatokat is megnézhetünk. Az eszköz rögzíti a művelet pontos paramétereit (pl. egy állomány megnyitása esetén milyen elérést kért az alkalmazás), vagy akár, hogy milyen hívások sorozatával jutott el ide az alkalmazás (azaz a verem tartalmát).



3. ábra: Esemény részletei

Egy konkrét lefutáson érdemes még megnézni a *Tools* menü összefoglaló listáit, a *File Summary* megmutatja például, hogy a futás során milyen fájlokat értünk el, és azokkal kapcsolatban mennyi esemény volt.

1.2 Egyszerű mintapélda vizsgálata

A gyakorlat keretében egy egyszerű, szándékosan hibás alkalmazást fogunk megvizsgálni.

1. Az alkalmazás forrása

Hozzunk létre egy `c:\demo` könyvtárat és benne egy `badapp.cs` fájlt a következő tartalommal.

```
using System;
using System.IO;

namespace BadApp
{
    class Program
    {
        private static readonly string dataFileLocation =
            Path.Combine(Environment.GetFolderPath(
                Environment.SpecialFolder.ProgramFilesX86),
                @"BME\BadApp\results.txt");

        public static void Main()
        {
            try
            {
                Console.WriteLine("Badly written application v1.3");
                Console.WriteLine("Processing...");

                System.Threading.Thread.Sleep(1000);

                using (StreamWriter outfile = new StreamWriter(dataFileLocation))
                {
                    outfile.Write(42);
                }

                Console.WriteLine("Processing done!");
            }
            catch
            {
                throw new Exception();
            }
        }
    }
}
```

Az alkalmazás kiír a konzolra, majd megpróbál írni egy fájlba, ami a *Program Files* könyvtáron belül van. (Figyelem: az alkalmazás szándékosan rossz, nem követendő példát mutat.)

2. Az alkalmazás lefordítása

Fordítsuk le a fenti programot! Nyissunk egy *Command Promptot*, majd hajtsuk végre a következő parancsokat.

```
cd \demo
c:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe badapp.cs
```

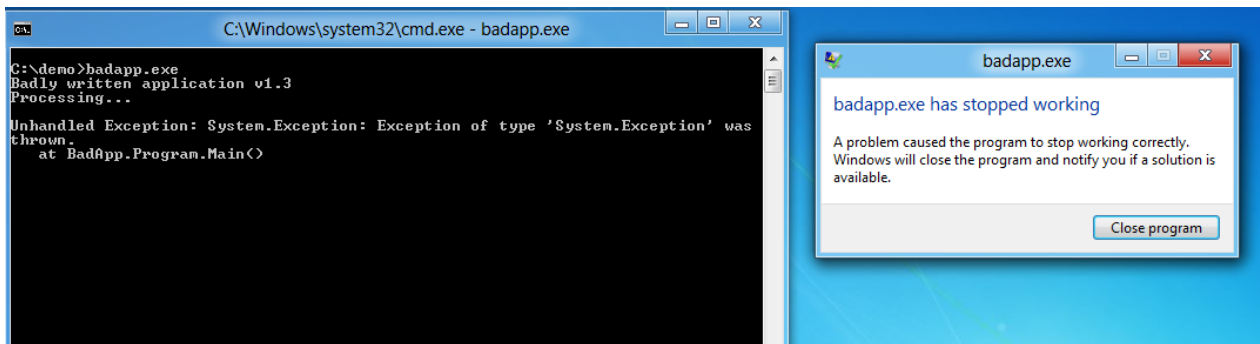
Attól függően, hogy milyen verziójú .NET Framework van telepítve a számítógépre, más lehet a C# fordító (*csc.exe*) elérési útja, ezt ha kell, akkor módosítsuk. A parancs eredményeképpen előáll a *badapp.exe* program.

3. Felkészülés a futásra

Hozzuk létre a *C:\Program Files (x86)\BME* könyvtárat (ehhez megerősítést kér majd a Windows, hisz alapesetben csak rendszergazdáknak van joga írni a *Program Files* könyvtárba).

4. Az alkalmazás futtatása

Futtassuk az alkalmazást egy nem rendszergazdai parancssorból. Ha minden „jól” megy, akkor hibát kell látnunk: egyrészt egy nem kezelt kivételt dob az alkalmazás, másrészt megjelenik a *Windows Error Reporting* ablaka.



4. ábra: A példa alkalmazás hibája

5. Az alkalmazás műveleteinek rögzítése

Indítsuk el a Process Monitor segédeszközt, és állítsunk be egy szűrőt, ami csak a *badapp.exe* műveleteit mutatja (szűrő: „processname”, „is”, „badapp.exe”, „include”).

The screenshot shows the Process Monitor application window with a list of operations performed by badapp.exe. The table below represents the data visible in the screenshot.

Time	Process Name	PID	Operation	Path	Result	Detail
15:05:...	badapp.exe	2076	Process Start		SUCCESS	Percent: PID: 1912
15:05:...	badapp.exe	2076	Thread Create		SUCCESS	Thread ID: 184
15:05:...	badapp.exe	2076	Load Image	C:\demo\badapp.exe	SUCCESS	
15:05:...	badapp.exe	2076	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7f1d9c0000, Image Size: 0x1b2000
15:05:...	badapp.exe	2076	CreateFile	C:\demo	SUCCESS	Desired Access: Execute/Traverse, Synchroniz...
15:05:...	badapp.exe	2076	CreateFile	C:\Windows\System32\vmcoree.dll	SUCCESS	Desired Access: Read Attributes, Disposition: Open...
15:05:...	badapp.exe	2076	QueryBasicInfor...	C:\Windows\System32\vmcoree.dll	SUCCESS	CreationTime: 2011.08.23. 23:16:58, LastAccessTim...
15:05:...	badapp.exe	2076	CloseFile	C:\Windows\System32\vmcoree.dll	SUCCESS	
15:05:...	badapp.exe	2076	CreateFile	C:\Windows\System32\vmcoree.dll	SUCCESS	Desired Access: Read Data/List Directory, Execute/...
15:05:...	badapp.exe	2076	CreateFileMapp...	C:\Windows\System32\vmcoree.dll	FILE LOCKED WITH ONLY R...	SyncType: SyncTypeCreateSection, PageProtection:
15:05:...	badapp.exe	2076	CreateFileMapp...	C:\Windows\System32\vmcoree.dll	SUCCESS	SyncType: SyncTypeOther
15:05:...	badapp.exe	2076	Load Image	C:\Windows\System32\vmcoree.dll	SUCCESS	Image Base: 0x7f11e80000, Image Size: 0x8f000
15:05:...	badapp.exe	2076	CloseFile	C:\Windows\System32\vmcoree.dll	SUCCESS	
15:05:...	badapp.exe	2076	Load Image	C:\Windows\System32\kernel32.dll	SUCCESS	Image Base: 0x7f1d880000, Image Size: 0x124000

5. ábra: badapp.exe műveletei

Indítsuk el ezután újra az alkalmazásunkat. Miután előjött a kivétel, állítsuk le a Process Monitorban a rögzítést (Ctrl+E). Ha minden jól megy, akkor 1000 körüli eseményt kellett rögzíteni (5. ábra).

6. A rögzített műveletek elemzése

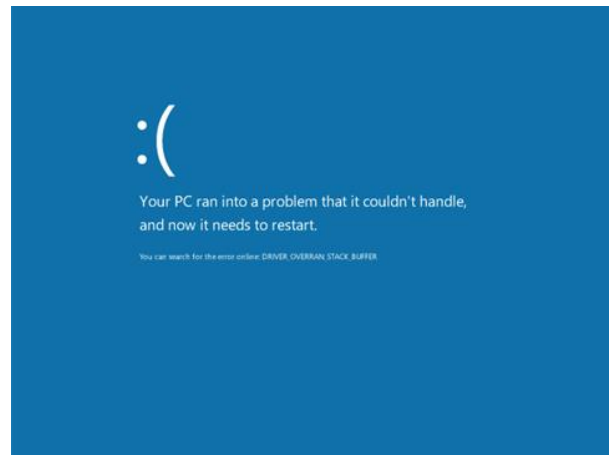
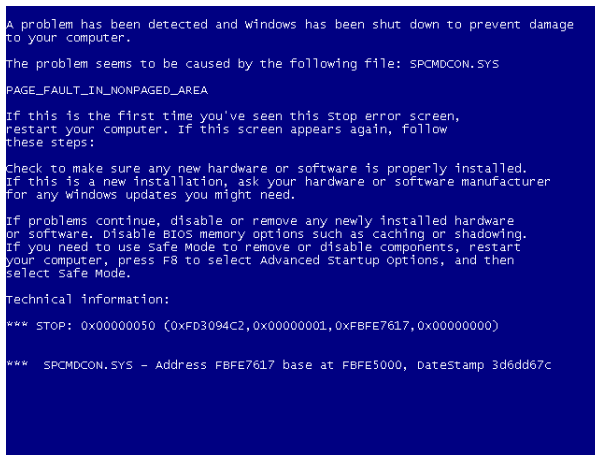
A rögzített műveletek alapján válaszoljunk a következő kérdésekre.

- Hány darab szál indult az alkalmazásban?
- Hány darab fájljal kapcsolatos műveletet végzett az alkalmazás?
- Milyen kiterjesztésű fájljal dolgozott az alkalmazás? (Használjuk a *File Summary* menüt.)
- Milyen visszatérési értékek szerepelnek a műveleteknél? (*Tools/Count Occurrences...*)
- Nézzük végig az első pár tíz műveletet. Mit csinál az indulása során az alkalmazás?
- Keressük meg a végén azt a részt, ahol a *Windows Error Reporting* komponenshez fordul, a hiba ez előtt nem sokkal keletkezett.
- Ha innen elkezdünk visszafelé lépkedni, akkor nemsokára megtaláljuk a hibát okozó műveletet, a results.txt fájlba akartunk írni, de azt megtagadta a rendszer. Milyen hozzáférési jogokat szeretett volna kapni az alkalmazás?

A feladat során egy egyszerű mintapéldán keresztül megnéztük, hogy hogyan lehet egy alkalmazás I/O műveleteit rögzíteni, majd abban eligazodni.

2 BSOD vizsgálata

Ha a Windows olyan hibát észlel kernel módban, amiből nem tud helyreállni, akkor a rendszer integritásának megóvása érdekében leállítja a futást, és a hírhedt „kék halál” (BSOD – Blue Screen of Death) képernyőt mutatja (6. ábra). A hiba típusát az úgynevezett STOP kód azonosítja, ehhez tartozik egy név is. Az alábbi ábra bal oldali része egy 0x50 kódú hibát mutat, ehhez a STOP kódhoz a PAGE_FAULT_IN_NONPAGED_AREA név tartozik. Windows 8 esetén már csak a nevet jeleníti meg, az ábrán látható példában ez DRIVER_OVERRAN_STACK_BUFFER. A STOP kódok részletes leírását az MSDN dokumentációjában találjuk [1].



6. ábra: „Kék halál” képernyő Windows 7 (bal) és Windows 8 (jobb) esetén

Ezt a feladatot különösen ajánlott nem a fizikai gépünkön, hanem virtuális gépben elvégezni!

1. Crash dump beállítása

A leállítás előtt a Windows készít egy úgynevezett crash dumpot, ami a memória tartalmának egy részét tartalmazza. Ennek részletességét a következő helyen állíthatjuk:

Computer / Properties / Advanced System Settings / Startup and Recovery / Settings / System Failure

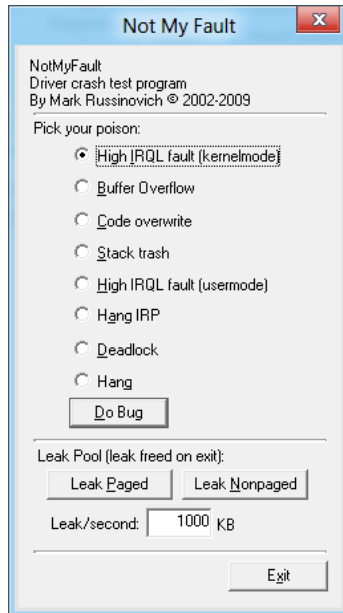
- a. Állítsuk be, hogy hiba esetén *Kernel memory dump*ot készítsen!
- b. Kapcsoljuk ki, hogy hiba esetén a rendszer automatikusan újrainduljon!

2. Hiba kiváltása

Kernel módú hiba kiváltására egy könnyen alkalmazható eszköz a NotMyFault példa eszköz¹. Indítsuk el rendszergazdaként az alkalmazást (arra figyeljünk, hogy ha 64 bites a virtuális gépünk, akkor az x64\Release könyvtárban lévő natív 64 bites verziót használjuk).

¹ Microsoft Press. Windows Internals, BookTools, <http://download.sysinternals.com/Files/Notmyfault.zip>

Okozzuk vele egy *High IRQL fault (kernelmode)* típusú hibát (7. ábra).



7. ábra: NotMy Fault segédprogram

Ha minden „jól” ment, akkor kék hiba képernyőt kell látnunk.

- a. Nézzük meg, hogy milyen információkat írt ki a képernyőre a rendszer. Csak ezekből mire tudnánk következtetni?
 - b. Indítsuk újra a virtuális gépet.
3. WinDbg beállítása

A crash dump elemzéséhez a WinDbg² eszközt fogjuk használni. Indítsuk is el rendszergazdaként (arra figyeljünk, hogy 64 bites operációs rendszer esetén a 64 bites verziót használjuk).

A vizsgálatok során az egyes modulok és az abban található függvények címeinek azonosításhoz úgynevezett symbol fájlokat használ a WinDbg, a szükséges symbol fájlokat képes online letölteni a Microsoft szerveréről.

Hozzuk létre a `c:\symbols` könyvtárat a virtuális gépen.

Állítsuk be WinDbg-ban a *Symbol File Path* beállítást (Ctrl+S) a következő értékre:

```
srv*c:\symbols*http://msdl.microsoft.com/download/symbols
```

4. Dump elemzése

- a. Nyissuk meg a dump fájlt (`C:\windows\memory.dmp`)!
- b. A dump fájl betöltésekor is már kiír jó pár információt a WinDbg. Keressük ki belőle, hogy mennyi ideig ment a hiba előtt a rendszer (uptime)!

² Microsoft. Windows Debugging Tools, <http://www.microsoft.com/whdc/devtools/debugging/>

- c. Mi a pontos STOP kód? Milyen paraméter értékei vannak? Nézzük meg az MSDN leírásban, hogy ez a kód esetén mit jelentenek ezek a paraméterek!
- d. Futtassuk le a `!analyze -v` parancsot! Ez elvégez egy alap analízist, ami a hiba körülményeit megvizsgálja. Nézzük meg, hogy milyen hívások sorozata vezetett a hibához (stack trace)! Milyen komponens okozta a hibát?

Ez az egyszerű elemzés az esetek nagy részében segít megállapítani a hibás eszközmeghajtót vagy komponenst.

A tanultak alapján próbáljunk egy ismeretlen dumpot is megvizsgálni. A tantárgy honlapjáról letölthető egy példa dump fájl, ebből mit tudunk megállapítani?

3 További információ

- [1] MSDN. Bug Check Codes, <http://msdn.microsoft.com/en-us/library/hh406232.aspx>
- [2] Microsoft. „How to read the small memory dump files that Windows creates for debugging”, KB 315263, URL: <http://support.microsoft.com/kb/315263/en-us>
- [3] MSDN. „Crash Dump Files”, URL: <http://msdn.microsoft.com/en-us/library/ff539316.aspx>