

Příloha A: Operační systémy Windows NT

Protože je téma mojí ročníkové práce úzce spjato s operačním systémem Windows, rozhodl jsem se začlenit do dokumentace i kapitolku, která popisuje základní principy a architekturu tohoto velmi známého a hojně využívaného operačního systému.

Základní principy

Windows patří mezi víceuživatelské operační systémy s preemptivním multitaskingem. Běžící aplikace jsou odděleny od vlastního jádra operačního systému. Kód jádra je vykonáván v privilegovaném módu procesoru (kernel mode – režim jádra), který umožňuje vykonávat privilegované instrukce a poskytuje přímý přístup k datům systému a k hardwaru. Kód aplikací (budu je nazývat též jako procesy) se vykonává v neprivilegovaném režimu procesoru (user mode – uživatelský mód) – procesy nemohou přímo přistupovat k hardwaru, vykonávat privilegované instrukce ani měnit (nebo číst) paměť jádra. Aplikace mohou využívat jen limitovaný počet rozhraní, které jim jádro nabízí. Pokud aplikace zavolá určité rozhraní, toto volání je zachyceno, procesor se přepne do režimu jádra a vykoná se potřebný kód. Poté se procesor vrátí zpátky do uživatelského režimu a řízení je opět předáno aplikaci.

Režim jádra se někdy nazývá Ring 0, uživatelský režim Ring 3. Ostatní Ring módy ve Windows nejsou použity. Princip je takový, že procesy běžící v Ring 3 se samy z bezpečnostních důvodů nemohou přepnout do režimu Ring 0. Kvůli některým zranitelnostem Windows NT je to i přesto možné, ale Microsoft neustále pracuje na jejich odstranění. Ve Windows 9x se vlastním přičiněním může do Ring 0 přepnout libovolný proces¹.

Jednotlivé součásti jádra Windows byly navrženy tak, aby neodporovaly principům objektově orientovaného programování. Každá součást jádra přistupuje k datům jiné součásti přes formální rozhraní, ačkoliv jí nic nebrání dostat se k datům přímo (všechny součásti jádra sdílejí stejný adresový prostor). Navzdory tomuto faktu nejsou Windows objektově orientovaným operačním systémem v pravém slova smyslu. Většina kódu je psána v jazyce C kvůli přenositelnosti a široké dostupnosti vývojových nástrojů. Jazyk C není objektově orientovaný, a tudíž nemohou být ani Windows.

Patří Windows mezi mikrokernelové operační systémy?

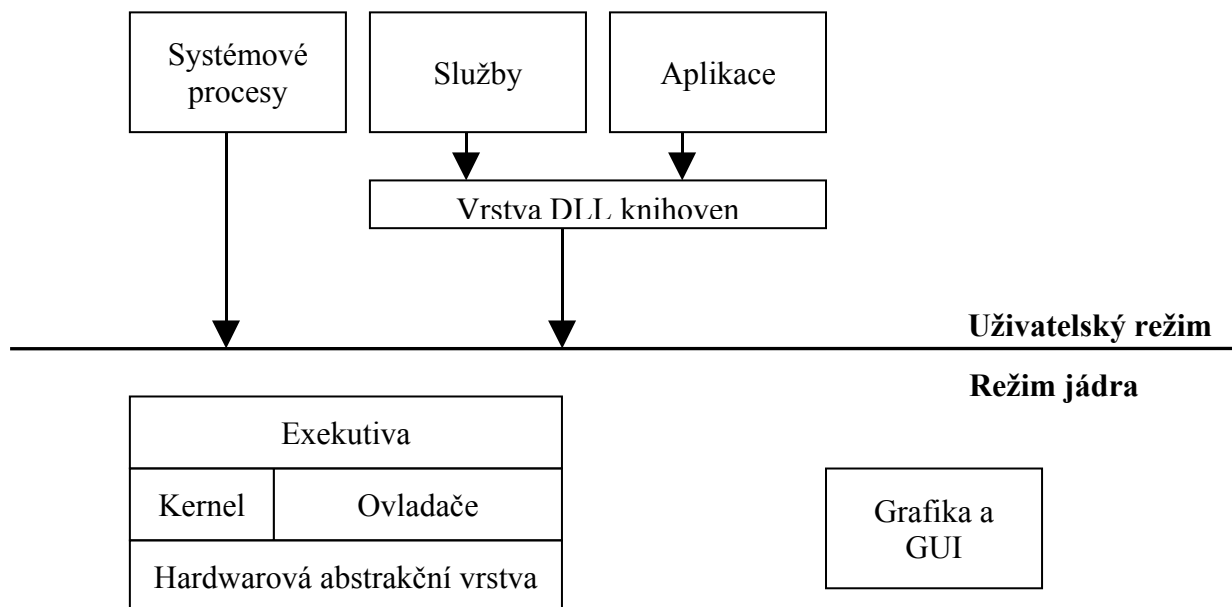
Ačkoliv to někteří tvrdí, Windows nepatří mezi mikrokernelové operační systémy. Mikrokernelové nazýváme systémy, které obsahují velmi malé jádro. Jádro poskytuje jen sadu základních rutin a všechny ostatní součásti (správce procesů, správce souborového systému) běží v uživatelském (neprivilegovaném) režimu procesoru. Výhodou mikrokernelových OS je bezesporu jejich stabilita. Pokud se například v kódu určité komponenty nachází chyba (například dojde ke čtení neplatné adresy paměti), nemusí nutně znamenat zhroucení celého

¹ Windows 9x mají tolik bezpečnostní děr, takže přepnutí do Ring 0 většinou není nutné

systemu, protože samotné jádro nenaruší. Pokud se však takováto chyba vyskytne v tak důležité komponentě jakou bezesporu je správce procesů nebo správce paměti, operační systém se nakonec zhroutí, ať je mikrokernelový nebo ne. To je jeden důvod, proč Windows nebyly programovány jako mikrokernel. Druhým důvodem je nižší rychlost mikrokernelových OS – procesor často musí přepínat mezi uživatelským a privilegovaným režimem. Protože Windows měly být využívány masami uživatelů, musely disponovat dostatečnou rychlostí, a tudíž si tolik přepínání nemohly dovolit.

Přehled hlavních komponent

Nyní máme základní povědomí o tom, jak Windows fungují. Podívejme se tedy na obrázek 1, který znázorňuje ty nejdůležitější součásti operačního systému.



Obrázek 1: Zjednodušený přehled komponent operačního systému Windows NT

Povšimněte si tlusté čáry. Nad touto čarou jsou znázorněny součásti, které běží v uživatelském režimu. Všechny jsou implementovány jako procesy a každý z nich běží uvnitř svého vlastního virtuálního adresového prostoru. Pod tlustou čarou jsou zobrazeny součásti běžící v režimu jádra. Jsou implementovány jako ovladače a sdílejí jeden adresový prostor.

Základními součástmi běžícími v uživatelském režimu jsou:

- **Systémové procesy**

Mezi systémové procesy patří například WINLOGON.EXE, Systém nebo správce služeb (services.exe). Pokud je některý ze systémových procesů neočekávaně ukončen, dojde k zastavení běhu systému a uživatel je informován modrou

obrazovkou (familiárně označovanou jako „modrá smrt“). Bez systémových procesů nemůže fungovat ani jádro operačního systému.

- **Služby**
Služby jsou procesy, které běží nezávisle na přihlašování a odhlašování uživatelů. Vykonávají svoji činnost na pozadí, od uživatele vstupy nepřijímají. Některé aplikace těchto výhod využívají a některé jejich součásti běží jako služby.
- **Aplikace**
Procesy běžící pod účtem přihlášeného uživatele. Jedná se o všechny známé programy jako Internet Explorer nebo Microsoft Word. Patří sem i staré programy určené pro MS-DOS. Při odhlášení uživatele jsou všechny aplikace běžící pod jeho účtem automaticky ukončeny.
- **Vrstva DLL knihoven**
DLL knihovny poskytují rozhraní, přes které musí aplikace komunikovat s jádrem operačního systému. Jejich úkolem je přeložit volání dokumentovaných rutin na volání rutin nedokumentovaných, které zavolají přímo jádro. Takovými DLL knihovnami prostředí Windows jsou například: kernel32.dll, user32.dll, gdi32.dll a advapi32.dll.

Jádro operačního systému se skládá z následujících komponent:

- **Exekutiva**
V exekutivě jsou implementovány základní služby operačního systému, například správa paměti, vláken a procesů, bezpečnost, vstupně-výstupní operace a komunikace mezi procesy.
- **Kernel**
Kernel obsahuje řadu nízkoúrovňových rutin sloužících k synchronizaci více procesorů, obsluze výjimek a přerušení a přidělování času procesoru jednotlivým vláknům.
- **Ovladače zařízení**
Mezi ovladače zařízení patří ovladače, které komunikují s periferními zařízeními, síťové ovladače a ovladače souborového systému.
- **Hardwarová abstrakční vrstva (HAL)**
HAL je téměř jediná součást Windows, která je závislá na hardware. Díky této vrstvě se ostatní součásti jádra nemusí starat například o rozdíly mezi různými základními deskami.
- **Grafika a GUI**
Tato součást se stará o uživatelské ovládací prvky a kreslení, zkrátka o celé grafické uživatelské rozhraní. Ve Windows NT starších než verze 4 tato komponenta běží v uživatelském režimu. Důvody pro přesun do režimu jádra jsou zřejmé: komponenta nyní může přímo komunikovat s ovladači a s dalšími součástmi jádra, což umožňuje rychleji provádět operace s grafikou.

Procesy a vlákna

Každý proces má svůj vlastní virtuální adresový prostor, který může využívat a nemůže přímo zasahovat do adresových prostorů jiných procesů. Samotný proces nevykonává žádný kód. Můžeme si jej představit jako kontejner, který obsahuje virtuální adresový prostor a vlákna, která vykonávají vlastní kód. Z toho vyplývá, že všechna vlákna jednoho procesu

sdílejí stejný adresový prostor. Adresový prostor sestává s bloků virtuální paměti, kterou si proces alokoval. Velikost virtuálního adresového prostoru procesu je 4 GB, může však využívat pouze spodní 2 GB (adresy menší než 0x80000000). Na adresách vyšších než 2 GB sídlí paměť jádra, jenž je pro něj nepřístupná.

Kód aplikací a programů je vykonáván prostřednictvím vláken. Každé vlákno charakterizuje struktura CONTEXT. CONTEXT je jediná struktura přístupná z uživatelského režimu, jejíž obsah je přímo závislý na hardwaru (architektuře procesoru). Na architektuře Intel x86 obsahuje hodnoty všech registrů procesoru. Když vlákno dostane čas na procesoru, je obsah struktury CONTEXT načten do registrů a po krátkou dobu se provádí jeho kód. Pak se opět hodnoty registrů uloží do struktury CONTEXT a procesor začne vykonávat kód jiného vlákna. Čas, který vlákno na procesoru stráví závisí na prioritě vlákna a prioritě procesu, do kterého patří.

Veškeré aplikace jsou uloženy na disku v tzv. PE souborech. PE (Portable executable) je název formátu spustitelného souboru. Mezi PE soubory patří většina souborů s příponami .exe, .dll, .sys, .vxd. Uvědomme si však, že to, zda je soubor spustitelný, nezávisí na jeho koncepcce, ale na jeho obsahu. Operační systém například sám od sebe nespustí žádný soubor s koncovkou .api², ale lze jej k tomu donutit. Existují samozřejmě i soubory, které nedodrží PE formát a jsou spustitelné – jedná se například o soubory s příponou .com a .bat.

PE formát je velmi složitý, tudíž si jej zde nebudeme dopodrobna popisovat. Podíváme se jen na některé struktury tohoto formátu, které budou důležité pro pochopení obsahu přílohy B. Jedná se o tyto tři části:

- **Tabulka importů (IAT – Import Address Table)**

V této struktuře se nachází seznam externích podprogramů, které kód uložený v PE souboru vyžaduje pro svoje provádění. Externí podprogramy se mohou nacházet v libovolném PE souboru na disku. Při vytváření procesu operační systém z této tabulky zjistí, které další moduly proces pro svůj běh potřebuje, a automaticky je zavede do virtuálního adresového prostoru.

- **Tabulka exportů**

Tabulka exportů se uplatňuje téměř výhradně u DLL knihoven, ale mohou ji obsahovat například i EXE soubory. Uchovává v sobě názvy a adresy podprogramů, které mohou být využívány jinými PE soubory. Například knihovna user32.dll dává světu k dispozici všechny podprogramy, které pracují s okny.

- **Vstupní bod (entry point)**

Když operační systém provede nutnou inicializaci modulu nově zaváděného do adresového prostoru procesu, předá řízení kódu uvnitř modulu na adresu, která se nazývá vstupní bod. Vstupní bod neobsahují jen EXE soubory, ale například i DLL knihovny. DLL knihovny na adresu vstupního bodu mohou umístit nejen inicializační kód, ale i ukončovací podprogram.

Více o PE souborech vědět nepotřebujeme.

Ovladače

² Koncovka .api je používána pluginy programu Adobe Acrobat Leader. Struktura pluginů se však od PE formátu příliš neliší.

Kód ovladačů je vykonáván v režimu jádra, proto patří k nejmocnějším programům běžícím na vašem počítači. Neexistují pro ně žádná bezpečnostní pravidla, ovladačům je dovoleno vše. Mohou číst a měnit libovolnou oblast paměti, mohou komunikovat přímo s hardwarem, dokonce pro ně není nemožné jej zničit. Na druhou stranu, napsat ovladač není jednoduché. Kód musí být totiž naprosto bez chyb. I malá chybička může mít fatální následky. Co skončilo u běžných aplikací chybovou hláškou, končí u ovladačů „modrou smrtí.“

Odlišnosti od procesů

Ovladače se od procesů liší v mnoha oblastech. Ovladač nemá svůj soukromí adresový prostor ani nevlastní žádná vlákna. Soubory všech ovladačů se nacházejí ve společném adresovém prostoru, který je namapován do každého procesu na adresy vyšší než 0x7FFFFFFF. Ovladač je vlastně jenom modul, něco jako DLL knihovna. Jeho kód je vykonáván vlákny různých procesů. Obecně mohou nastat tyto situace:

- Proces zaslal ovladači požadavek (zprávu). Kód zpracování požadavku a vygenerování odpovědi vykonává vlákno procesu, které požadavek zaslalo.
- Kód ovladače je vykonáván v kontextu nějakého vlákna procesu System. To se děje například při volání inicializační rutiny nebo při vykonávání kódu některých ovladačem vytvořených vláken.
- Kód může být vykonáván v rámci kontextu náhodného procesu a vlákna. Tato situace nastane, pokud je kód volán v rámci obsluhy přerušení. Ovladač běží v kontextu vlákna, které právě dostalo čas na procesoru. Tento případ nastává u driverů komunikujících s hardwarem.

Většina ovladačů nevyvíjí nezávislou činnost jako normální procesy. Ovladač při své inicializaci obvykle zaregistruje některé rutiny, které budou volány při určitých událostech. Mezi takové události patří požadavek zaslaný procesem, obsluha přerušení nebo vypínání počítače (prováděné standardním způsobem).

Další zajímavou odlišností od procesů je využívání paměti. Ovladače totiž mohou využívat jak paměť stránkovanou, tak i nestránkovanou. Obsah nestránkované paměti není nikdy uložen do stránkovacího souboru a je kdykoliv dostupný. Pokud by se nějaká rutina vykonávaná s vysokou prioritou pokusila přistoupit k paměti odložené na disku, nemusel by být operační systém schopen potřebné stránky umístit do fyzické paměti. Využívat nestránkovanou paměť normálními procesy je sice možné, ale v drtivé většině případů zbytečné.

Priority

Kód výše zmiňovaných rutin je vykonáván s různou prioritou. U procesů priorita není až tak významná vlastnost, u ovladačů na ní záleží velmi mnoho a často vývoj ovladače velmi komplikuje. Určité služby operačního systému mohou být volány jen při určité prioritě. Obecně platí, že kód s vyšší prioritou může použít méně služeb než kód s prioritou nižší. Kód s prioritou A nemůže být přerušen kódem s prioritou $< A$. Prioritám se někdy říká *hardwarové priority* a souhrnně se pro ně používá zkratka IRQL. Procesy mohou svoji prioritu libovolně měnit, ovladače ne. Rutina ovladače může svoji prioritu pouze zvýšit, ale

ještě před svým skončením ji musí nastavit na původní hodnotu. Priorita je určena číslem. Podívejme se na některé možné hodnoty:

- **PASSIVE_LEVEL**
Tato priorita je nejnižší. Vlákno s touto může být přerušeno jiným vláknem. Dostupné jsou téměř všechny služby operačního systému. Tuto prioritu mají vlákna procesu System a je s ní vykonáván například inicializační a ukončovací podprogram ovladače.
- **DISPATCH_LEVEL**
Při této prioritě již nelze využívat mnoha systémových volání. Nelze pracovat se soubory a není bezpečné přistupovat k stránkované paměti. Kód s touto prioritou může být přerušen jedině hardwarovými přerušeními. Procesor, který vykonává takový kód, neprovádí multitasking, dokud se priorita nesníží. Proto je důrazně doporučováno provádět při této prioritě co nejméně operací.
- **DIRQL**
S touto prioritou (a vyšší) jsou obsluhována hardwarová přerušení. Obslužné rutiny mohou jen omezeně pracovat s nestránkovanou pamětí a jejich kód by měl být velmi Krátký, aby zbytečně nezatěžoval procesor. Takové rutiny většinou získaná data předávají ke zpracování kódu běžícímu s nižší prioritou, protože jejich možnosti jsou v tomto ohledu velmi omezené.

Instalace, spuštění a uvolnění z paměti

Pro instalaci ovladače je nutné zapsat několik položek do registru Windows³. Tyto položky systému říkají, jak se ovladač jmenuje, kde je umístěn jeho soubor a za jakým podmínek má být spuštěn. Ovladač může být spuštěn buď během bootovací sekvence, nebo na příkaz určitého procesu. Proces nemusí „ručně“ vyplňovat tyto hodnoty do Registru, pro instalaci ovladači stačí zavolat jedinou API funkci, která vše potřebné vykoná⁴.

Po instalaci a spuštění se provede inicializační rutina, která ovladač připraví pro vykonávání daného úkolu. Ovladač pak plní svůj úkol, dokud není uvolněn z paměti. Těsně před uvolněním z paměti je volána ukončovací rutina, jejíž úkolem je po ovladači uklidit veškerý nepořádek.

Uvolnění z paměti však nemusí být vůbec snadné. Pokud totiž ovladač v sobě neobsahuje ukončovací rutinu, standardní metodou jej uvolnit nelze. Nejjednodušší možností je smazání položek registru, které tam byly zapsány při instalaci následované restartováním počítače. Některé ovladače (zejména součásti malwaru) se budou snažit svoje položky v registru chránit, a tak jejich smazání (položek) může být velmi obtížné. Pak už zbývá téměř jediná možnost – pokusit se položky smazat v nouzovém režimu, kdy možná ovladač nebude aktivní.

Objekty jádra a jejich sdílení

³ Položky ovladačů se zapisují do klíče HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services

⁴ Bohužel existují nedokumentované postupy, které umožňují načíst ovladač do paměti jádra bez nutnosti zápisu do registru.

Když nějaký proces například otevře jiným procesem nepoužívaný soubor, jádro operačního systému alokuje oblast paměti, kam uloží informace o právě otevřeném souboru – jádro systému vytvoří tzv. *objekt jádra*. Pokud jiný proces otevře stejný soubor, k alokaci paměti nedochází – příslušnému objektu jádra se jen zvýší počet referencí. Pokud dojde k zavření souboru, počet referencí klesne o 1. Dosáhne-li počet referencí nuly, příslušný objekt jádra už není žádným procesem využíván a paměť, ve které se nachází, je uvolněna. Tento mechanismus umožňuje efektivní sdílení právě využívaných objektů a tvoří základ pro komunikaci mezi procesy. Nemusíme však o tomto mechanismu vědět, protože je vše implementováno interně a procesy běžící v uživatelském režimu do toho „nevidí.“ Ovladače mohou počet referencí určitého objektu volně upravovat – ovladače musí také využívat objektů jádra, chtějí-li například zapisovat do souboru. Manipulací s počtem referencí může ovladač zajistit, aby objekt zůstal v paměti, i když není žádným procesem využíván.

Když proces otevře soubor, operační systém mu přidělí tzv. *handle*. Toto handle je možné použít při volání různých funkcí Windows API pracujících se soubory. Je to odkaz na určitý objekt jádra. Není to však adresa objektu v paměti, ale pouze číslo. Z hodnoty tohoto čísla proces nemůže zjistit téměř nic. Bez použití funkcí Windows API nelze zjistit nic o objektu, který určité handle reprezentuje. Pro jádro operačního systému hodnota handle význam samozřejmě má. Hodnota handle je unikátní pro každý proces.

Pro každý proces si jádro pamatuje, které objekty využívá. Tyto informace jsou uloženy ve struktuře s názvem *tabulka handleů* (handle table). Tabulka handleů je pole. Hodnota handle je index položky, kterou handle reprezentuje.

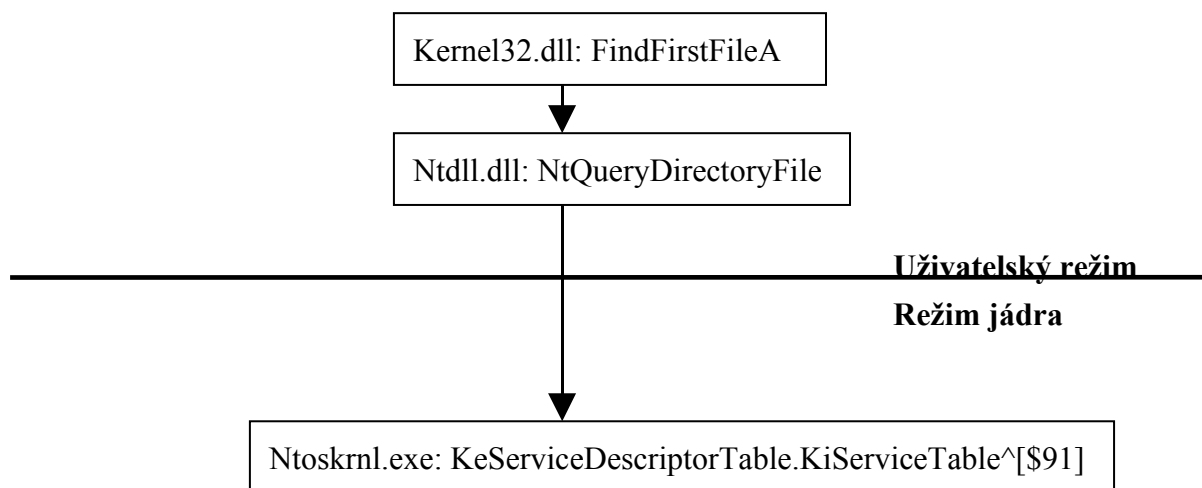
Objektů jádra je mnoho druhů, ale pro jednoduchost jsme zde mluvili jenom o souborech.

Důležité struktury systému Windows NT

V paměti jádra se nachází několik velmi významných struktur, které jsou využívány například při obsluze přerušení nebo když proces běžící v uživatelském režimu volá jádro systému. V této kapitole se s dvěma takovými strukturami letmo seznámíme a v druhé příloze se dozvíte další podrobnosti. Níže popisované struktury jsou totiž často využívané nejen rootkity, ale i různými bezpečnostními programy.

SSDT

SSDT je zkratka anglických slov *Systém Service Descriptor Table* a v této dokumentaci ji také nazývám „tabulka interních služeb systému.“ Tato struktura je použita, pokud proces v uživatelském režimu volá některou ze služeb jádra. Adresa SSDT je exportována hlavním modulem jádra (většinou *ntoskrnl.exe*) pod jménem *KeServiceDescriptorTable*. Ve svých položkách obsahuje adresu struktury *Systém Service Dispatch Table*, která se zkráceně nazývá *KiServiceTable* (ale někdy se používá i zkratka *SDT*). *KiServiceTable* je pole ukazatelů. Obsahuje adresy všech služeb jádra, které jsou přístupné z uživatelského režimu. Právě řečená teorie může být pro některé těžko stravitelná – ukážeme tedy vše na příkladu.



Obrázek 2: Průběh volání funkce `FindFirstFileA`

Na obrázku vidíme průběh volání funkce Windows API `FindFirstFileA`, která se používá při zjišťování souborů v adresáři. Tato funkce je exportována knihovnou `Kernel32.dll`. Pokud proces tuto rutinu zavolá, v knihovně `Kernel32.dll` dojde k volání podprogramu `NtQueryDirectoryFile` z knihovny `ntdll.dll`. Tento podprogram je velice krátký. Jeho úkolem je přeložit název volané funkce na index do pole `KiServiceTable`. Číslo služby `NtQueryDirectoryFile` je `0x91`. Program toto číslo uloží do registru `EAX`, do registru `EDX` je uložen vrchol zásobníku, kde se nacházejí parametry. Následuje instrukce `SYSENTER` (na starších systémech volání přerušeni `0x2E`). Tím dojde k přechodu do režimu jádra a řízení je předáno rutině, která obsluží vzniklé přerušeni. Rutina přkopíruje parametry do paměti jádra (využije přitom obsah registru `EDX`). Ze `SSDT` je zjištěna adresa pole `KiServiceTable`. Nakonec rutina předá řízení na adresu, která je uložena v tomto poli pod indexem v registru `EAX`.

Z předchozího popisu vyplývá důležitost `SSDT` a `KiServiceTable`. Jsou zodpovědné za předání volání z uživatelského režimu správné službě jádra.

IDT

IDT je zkratka ze slov *Interrupt Descriptor Table*. Do češtiny toto sousloví překládáme jako „tabulka vektorů přerušeni.“ Tato struktura obsahuje adresy obslužných rutin

pro všechna hardwarová i softwarová přerušení. Na obsluhu přerušení stojí základní principy operačního systému. Přerušení umožňují implementovat multitasking, komunikaci s hardwarem a mnoho dalších mechanismů by bez nich nefungovalo. Určité přerušení je generováno například při každém stisku nebo uvolnění klávesy. Určitá přerušení jsou generována, dojde-li k chybě (dělení nulou, výpadek stránky, přehřívání procesoru). Tabulka vektorů přerušení je zkrátka velmi důležitá.

Jak to ale funguje? Velmi jednoduše. Když je vyvoláno přerušení, procesor uloží právě rozdělanou práci a přerušení obslouží – předá řízení rutině uvedené v tabulce přerušení. Po skončení rutiny se procesor opět vrátí k původní práci.

Přerušení je celkem 256. Některá jsou využívána hardwarem (například klávesnicí), některá jsou softwarová (například přerušení 0x2E zmíněné v minulé kapitole). Některá přerušení nejsou standardně využita a může jich použít libovolný ovladač.

Adresa tabulky vektorů přerušení je zapsána ve struktuře IDTINFO, jejíž umístění lze získat instrukcí SIDT. Každý procesor má vlastní tabulku přerušení, takže instrukce SIDT vrátí adresu struktury IDTINFO procesoru, kterým byla vykonána.