Microsoft

Windows od środka

Architektura systemu, procesy, wątki, zarządzanie pamięcią i dużo więcej

Wydanie VII

Professional

Pavel Yosifovich
Alex Ionescu
Mark E. Russinovich
David A. Solomon

Helion

Tytuł oryginału: Windows Internals, Part 1: System architecture, processes, threads, memory management, and more (7th Edition)

Tłumaczenie: Piotr Pilch (wstęp, rozdz. 1, 3 – 6), Zbigniew Waśko (rozdz. 2, 7)

ISBN: 978-83-283-3901-9

Authorized translation from the English language edition, entitled: WINDOWS INTERNALS, PART l: SYSTEM ARCHITECTURE, PROCESSES, THREADS, MEMORY MANAGEMENT, AND MORE, Seventh Edition; ISBN 0735684189; by Pavel Yosifovich; and by Alex Ionescu; and by Mark E. Russinovich; and David A. Solomon; published by Pearson Education, Inc, publishing as Microsoft Press. Copyright © 2017 by Pavel Yosifovich, Alex Ionescu, Mark E. Russinovich and David A. Solomon

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Spis treści

ROZDZIAŁ 1.

Zagadnienia i narzędzia

W rozdziale zostaną zaprezentowane zagadnienia i terminy związane z systemem operacyjnym Microsoft Windows, które będą wykorzystywane w książce, takie jak interfejs API systemu Windows, procesy, wątki, pamięć wirtualna, tryb jądra i tryb użytkownika, obiekty, dojścia, zabezpieczenia i rejestr. Zostaną również przedstawione narzędzia, których możesz użyć do poznawania wewnętrznych mechanizmów systemu Windows, takie jak debugger jądra, Monitor wydajności oraz kluczowe narzędzia oferowane przez witrynę Windows Sysinternals (*http://www.microsoft.com/technet/sysinternals*). Poza tym wyjaśnimy, jak zastosować zestawy Windows Driver Kit (WDK) oraz Windows Software Development Kit (SDK) w roli zasobów służących do znajdowania dodatkowych informacji o wewnętrznych mechanizmach systemu Windows.

Zadbaj o zrozumienie wszystkiego, co zaprezentowano w tym rozdziale, ponieważ w przypadku reszty książki założono, że tak będzie.

Wersje systemu operacyjnego Windows

W książce omówiono najnowsze wersje systemów operacyjnych Microsoft Windows przeznaczonych dla klientów i serwerów, a mianowicie system Windows 10 (wersja 32-bitowa dla procesorów x86 i ARM oraz wersja 64-bitowa dla procesorów x64) oraz system Windows Server 2012 R2 (dostępny tylko w wersji 64-bitowej). Jeśli nie zostało to wyraźnie określone, treść książki dotyczy wszystkich tych wersji. W roli podstawowych informacji w tabeli 1.1 wyszczególniono nazwy systemów Windows wraz z numerem wewnętrznym ich wersji oraz datą opublikowania.

Począwszy od systemu Windows 7, numery wersji wydają się schodzić z dobrze zdefiniowanej drogi. W jego przypadku numer wersji to 6.1, a nie 7. Gdy w momencie pojawienia się systemu Windows Vista numer wersji podniesiono na 6.0, z powodu popularności systemu Windows XP niektóre aplikacje nie były w stanie wykryć poprawnego systemu operacyjnego, ponieważ projektanci zastosowali sprawdzanie głównego numeru większego lub równego 5 oraz dodatkowego numeru większego lub równego 1, co w przypadku systemu Windows Vista nie było spełnione. Wyciągając z tego lekcję, w celu zminimalizowania takich niezgodności firma Microsoft zdecydowała się określić główny numer wersji jako 6 oraz dodatkowy numer wersji jako 2 (większe niż 1). W wypadku systemu Windows 10 numer wersji został jednak uaktualniony do 10.0.

TABELA 1.1. *Wersje systemu operacyjnego Windows*

Nazwa produktu	Numer wewnętrzny wersji	Data opublikowania
Windows NT 3.1	3.1	Lipiec 1993 r.
Windows NT 3.5	3.5	Wrzesień 1994 r.
Windows NT 3.51	3.51	Maj 1995 r.
Windows NT 4.0	4.0	Lipiec 1996 r.
Windows 2000	5.0	Grudzień 1999 r.
Windows XP	5.1	Sierpień 2001 r.
Windows Server 2003	5.2	Marzec 2003 r.
Windows Server 2003 R2	5.2	Grudzień 2005 r.
Windows Vista	6.0	Styczeń 2007 r.
Windows Server 2008	6.0 (Service Pack 1)	Marzec 2008 r.
Windows 7	6.1	Październik 2009 r.
Windows Server 2008 R2	6.1	Październik 2009 r.
Windows 8	6.2	Październik 2012 r.
Windows Server 2012	6.2	Październik 2012 r.
Windows 8.1	6.3	Październik 2013 r.
Windows Server 2012 R2	6.3	Październik 2013 r.
Windows 10	10.0 (kompilacja 10240)	Lipiec 2015 r.
Windows 10 version 1511	10.0 (kompilacja 10586)	Listopad 2015 r.
Windows 10 version 1607 (Anniversary Update)	10.0 (kompilacja 14393)	Lipiec 2016 r.
Windows Server 2016	10.0 (kompilacja 14393)	Październik 2016 r.

Uwaga. Począwszy od systemu Windows 8, funkcja API `GetVersionEx` systemu Windows zwraca domyślnie jego numer wersji jako 6.2 (Windows 8) niezależnie od faktycznie używanego systemu operacyjnego (funkcja ta jest też określona jako niezalecana). Ma to na celu zminimalizowanie problemów ze zgodnością, ale również jest to wskaźnik, że w większości sytuacji sprawdzanie wersji systemu nie jest najlepszym rozwiązaniem. Wynika to z tego, że niektóre komponenty mogą być instalowane niestandardowo bez zgodności z oficjalną wersją systemu Windows. Jeśli jednak wymagasz informacji o faktycznej wersji systemu operacyjnego, możesz ją uzyskać za pomocą funkcji `VerifyVersionInfo` lub nowszych pomocniczych interfejsów API wersji, takich jak `IsWindows8OrGreater`, `IsWindows8Point1OrGreater`, `IsWindows10OrGreater` i `IsWindowsServer` oraz innych podobnych. Dodatkowo zgodność z systemem operacyjnym może zostać wskazana w manifeście pliku wykonywalnego, co powoduje zmianę wyników działania funkcji `VerifyVersionInfo`.

Informacje o wersji systemu Windows możesz wyświetlić za pomocą narzędzia ver działającego w trybie wiersza poleceń lub przy użyciu narzędzia winver z interfejsem graficznym. Poniżej zaprezentowano zrzut ekranu narzędzia winver uruchomionego w systemie Windows 10 Home (wersja 1607).

Powyższe okno zawiera też numer kompilacji systemu Windows (w przykładzie 14393.1358), który może być przydatny dla uczestników programu Windows Insider (osób, które zarejestrowały się w celu uzyskania możliwości zaznajomienia się z wcześniejszymi wersjami testowymi systemu Windows). Numer kompilacji może być też pomocny przy zarządzaniu aktualizacjami zabezpieczeń, ponieważ informuje o tym, jaki zainstalowano poziom poprawek.

Windows 10 i przyszłe wersje systemu Windows

W przypadku systemu Windows 10 firma Microsoft zadeklarowała, że będzie aktualizować system Windows szybciej, niż następowało to wcześniej. Nie pojawi się oficjalnie Windows 11. Zamiast tego w ramach usługi Windows Update (lub innego modelu usług dla przedsiębiorstw) istniejący system Windows 10 zostanie zaktualizowany do nowszej wersji. Gdy pisano książkę, nastąpiły dwie takie aktualizacje: w listopadzie 2015 r. (aktualizacja znana jako *wersja 1511*, odwołująca się do roku i miesiąca udostępnienia) oraz w lipcu 2016 r. (*wersja 1607*, która jest znana również pod marketingową nazwą *Anniversary Update*).

Uwaga. Wewnętrznie firma Microsoft w dalszym ciągu falowo tworzy wersje systemu Windows. Na przykład początkowa wersja systemu Windows 10 miała nazwę kodową *Threshold 1*, natomiast aktualizacja z listopada 2015 r. nosiła nazwę *Threshold 2*. Trzy kolejne fazy aktualizacji mają nazwy *Redstone 1* (wersja 1607), *Redstone 2* i *Redstone 3*.

Windows 10 i platforma OneCore

Przez lata zostało rozwiniętych kilka wariantów systemu Windows. Oprócz najpopularniejszego systemu Windows działającego na komputerach PC, dostępna jest konsola do gier Xbox 360 z odmianą systemu Windows 2000. Urządzenie Windows Phone 7 wyposażone jest w wariant oparty na systemie Windows CE (system operacyjny czasu rzeczywistego firmy Microsoft). Utrzymywanie i rozszerzanie wszystkich tych baz kodu jest oczywiście utrudnione. W związku z tym firma Microsoft zdecydowała się na połączenie w jedno jąder oraz pomocniczych plików binarnych podstawowej platformy. Proces ten rozpoczął się w przypadku systemów Windows 8 i Windows Phone 8, które korzystają ze wspólnego jądra (systemy Windows 8.1 i Windows Phone 8.1 mają jednakowy interfejs API środowiska uruchomieniowego systemu Windows). Proces łączenia zakończono w systemie Windows 10. Uzyskana wspólna platforma nosi nazwę OneCore. Działa ona na komputerach PC, telefonach, konsoli do gier Xbox One oraz na urządzeniach HoloLens i Internet of Things (IoT), takich jak Raspberry Pi 2.

Oczywiście wszystkie te rodzaje urządzeń różnią się znacząco od siebie. W wypadku części z nich niektóre funkcje są po prostu niedostępne. Na przykład obsługa myszy lub fizycznej klawiatury w przypadku urządzenia HoloLens nie ma sensu, dlatego nie możesz oczekiwać obecności odpowiednich komponentów w wersji systemu Windows 10 przeznaczonej dla tego urządzenia. Jednakże jądro, sterowniki i binaria podstawowej platformy są zasadniczo takie same (wraz z ustawieniami opartymi na rejestrze i/lub zasadach, które mają sens ze względu na wydajność lub z innych powodów). Tego rodzaju przykładowa zasada zostanie zaprezentowana w punkcie „Mechanizm API Sets” rozdziału 3., „Procesy i zadania”.

W książce obszernie omówiono wewnętrzne mechanizmy jądra platformy OneCore, niezależnie od tego, na jakim urządzeniu działa. Eksperymenty opisane w książce koncentrują się jednak na komputerze stacjonarnym wyposażonym w mysz i klawiaturę (głównie ze względu na wygodę), ponieważ nie jest łatwe (a czasami oficjalnie niemożliwe) przeprowadzenie doświadczeń w przypadku innych urządzeń, takich jak telefony lub konsola Xbox One.

Podstawowe pojęcia i terminy

W poniższych punktach dokonano wprowadzenia do najbardziej fundamentalnych pojęć obecnych w systemie Windows, które mają kluczowe znaczenie dla zagadnień omówionych w pozostałej części książki. Wiele pojęć, takich jak procesy, wątki i pamięć wirtualna, objaśniono szczegółowo w kolejnych rozdziałach.

Interfejs API systemu Windows

Interfejs programowania aplikacji API (ang. *Application Programming Interface*) systemu Windows to systemowy interfejs programowania trybu użytkownika zapewniający dostęp do rodziny systemów operacyjnych Windows. Przed wprowadzeniem 64-bitowych wersji systemu Windows interfejs programowania 32-bitowych wersji tego systemu nosił nazwę *Win32 API*, aby odróżnić go od oryginalnego 16-bitowego interfejsu API systemu Windows, który był interfejsem programowania pierwotnych 16-bitowych wersji systemu Windows. W książce termin *interfejs API systemu Windows* odnosi się zarówno do 32-bitowych, jak i 64-bitowych interfejsów programowania.

Uwaga. Czasami w miejsce terminu *interfejs API systemu Windows* używamy terminu *Win32 API*. Oba terminy odnoszą się jednak do wariantów 32- i 64-bitowych.

Uwaga. Interfejs API systemu Windows opisano w dokumentacji zestawu SDK systemu Windows (zajrzyj do punktu „Windows Software Development Kit" zamieszczonego w dalszej części rozdziału). Dokumentacja jest dostępna bezpłatnie pod adresem *https://developer.microsoft.com/en-us/windows/desktop/develop*. Dołączono ją również do wszystkich poziomów subskrypcji w portalu Microsoft Developer Network (MSDN), z którym związany jest program wsparcia projektantów aplikacji. Znakomity opis tego, jak utworzyć podstawowy interfejs API systemu Windows, zamieszczono w książce *Windows via C/C++. Fifth Edition* napisanej przez Jeffreya Richtera i Christophe'a Nasarre'a (Microsoft Press, 2007 r.).

Warianty interfejsu API systemu Windows

Pierwotnie interfejs API systemu Windows składał się wyłącznie z funkcji napisanych w stylu języka C. Obecnie projektanci mogą skorzystać z tysięcy takich funkcji. Język C był naturalnym wyborem w początkach istnienia systemu Windows, ponieważ stanowił najmniejszy wspólny mianownik (oznacza to, że kod napisany w tym języku był dostępny również z poziomu innych języków), a ponadto był na tyle niskopoziomowy, aby udostępniać usługi systemu operacyjnego. Wadą była niewielka liczba funkcji oraz brak spójności nazewniczej i logicznego grupowania (na przykład przestrzenie nazw w języku C++). Rezultatem tych trudności było utworzenie kilku nowszych interfejsów API korzystających z innego mechanizmu API, czyli modelu COM (ang. *Component Object Model*).

Model COM został pierwotnie zaprojektowany w celu umożliwienia aplikacjom pakietu Microsoft Office komunikacji i wymiany danych między dokumentami (na przykład osadzanie wykresu Excela w dokumencie Worda lub prezentacji programu PowerPoint). Taka możliwość nosi nazwę funkcji OLE (ang. *Object Linking and Embedding*). Oryginalnie funkcja ta została zaimplementowana za pomocą starego mechanizmu przesyłania komunikatów systemu Windows o nazwie *Dynamic Data Exchange* (DDE). Mechanizm ten z natury był ograniczony, dlatego opracowano nową metodę komunikacji, czyli model COM. Tak naprawdę, początkowo w momencie opublikowania około 1993 r. miał on nazwę OLE 2.

Model COM oparty jest na dwóch podstawowych zasadach. Po pierwsze, klienty komunikują się z obiektami (czasami nazywanymi obiektami serwerowymi COM) za pośrednictwem interfejsów, czyli odpowiednio zdefiniowanych kontraktów z zestawem logicznie powiązanych metod grupowanych w ramach mechanizmu rozsyłania tabeli wirtualnej, który w wypadku kompilatorów języka C++ stanowi też typowy sposób implementowania rozsyłania funkcji wirtualnych. Efektem tego jest zgodność binarna oraz usunięcie problemów z przekłamywaniem nazw kompilatora. W konsekwencji możliwe jest wywołanie tych metod w wielu językach (i kompilatorach), takich jak C, C++, Visual Basic, .NET, Delphi oraz innych. Po drugie, implementacja komponentów ładowana jest dynamicznie zamiast statycznego powiązania jej z klientem.

Termin *serwer modelu COM* odnosi się zwykle do biblioteki Dynamic Link Library (DLL) lub pliku wykonywalnego (EXE), gdzie implementowane są klasy modelu COM. Model ten oferuje inne ważne funkcje powiązane z zabezpieczeniami, marshallingiem realizowanym między procesami,

modelem wątkowości itp. Obszerne omówienie modelu COM wykracza poza zakres książki. Znakomitą prezentację tego modelu można znaleźć w książce *Essential COM* autorstwa Dona Boxa (Addison-Wesley, 1998 r.).

Uwaga. Przykłady interfejsów API dostępnych za pośrednictwem modelu COM obejmują DirectShow, Windows Media Foundation, DirectX, DirectComposition, Windows Imaging Component (WIC) oraz Background Intelligent Transfer Service (BITS).

Architektura Windows Runtime

W systemie Windows 8 wprowadzono nowy interfejs API oraz wspierające środowisko uruchomieniowe o nazwie *Windows Runtime* (określane czasem za pomocą skrótu *WinRT*, którego nie należy mylić z wycofaną wersją Windows RT systemu operacyjnego Windows opartego na architekturze ARM). Architektura Windows Runtime składa się z usług platformy kierowanych szczególnie do projektantów tak zwanych aplikacji *Windows Apps* (znanych wcześniej pod nazwami *Metro Apps*, *Modern Apps*, *Immersive Apps* i *Windows Store Apps*). Aplikacje Windows Apps mogą być przeznaczone dla wielu wariantów urządzeń, począwszy od niewielkich urządzeń IoT i telefonów, a skończywszy na tabletach, laptopach i komputerach stacjonarnych, a nawet takich urządzeniach, jak konsola Xbox One i Microsoft HoloLens.

Z perspektywy interfejsu API architektura WinRT zbudowana jest na bazie modelu COM, poszerzając jego podstawową infrastrukturę o różne rozszerzenia. Na przykład w architekturze tej dostępne są kompletne metadane typów (przechowywane w plikach WINMD oraz oparte na formacie metadanych środowiska .NET), które rozszerzają podobne pojęcie obecne w modelu COM znane jako biblioteki typów. Z punktu widzenia projektu interfejsów API architektura WinRT jest znacznie bardziej spójna niż klasyczne funkcje API systemu Windows, oferując hierarchie przestrzeni nazw, jednolite nazewnictwo oraz wzorce programistyczne.

W przeciwieństwie do zwykłych aplikacji systemu Windows (nazywanych obecnie *aplikacjami dla komputerów stacjonarnych z systemem Windows* lub *klasycznymi aplikacjami systemu Windows*), aplikacje Windows Apps podlegają nowym regułom.

Relacja między różnymi interfejsami API i aplikacjami nie jest bezpośrednia. Aplikacje dla komputerów stacjonarnych mogą korzystać z podzbioru interfejsów API architektury WinRT. Z kolei aplikacje Windows Apps mogą używać podzbioru interfejsów API architektury Win32 i modelu COM. W dokumentacji portalu MSDN znajdziesz szczegóły dotyczące tego, jakie interfejsy API są dostępne z poziomu każdej platformy aplikacji. Zauważ jednak, że na podstawowym poziomie binarnym interfejs API architektury WinRT w dalszym ciągu bazuje na starszych binariach i interfejsach API systemu Windows, nawet pomimo tego, że dostępność określonych interfejsów API może nie być udokumentowana lub wspierana. Nie jest to nowy natywny interfejs API systemu, lecz podobnie jak w przypadku środowiska .NET, nadal korzysta on z tradycyjnego interfejsu API systemu Windows.

Aplikacje napisane w językach C++, C# (lub innych językach .NET) oraz JavaScript mogą z łatwością stosować interfejsy API architektury WinRT, co zawdzięczają projekcjom językowym opracowanym z myślą o tej architekturze. W wypadku języka C++ firma Microsoft utworzyła niestandardowe rozszerzenie o nazwie C++/CX, które upraszcza korzystanie z typów architektury WinRT. Zwykła warstwa współdziałania modelu COM przeznaczona dla środowiska .NET (z kilkoma wspierającymi rozszerzeniami uruchomieniowymi) umożliwia dowolnemu językowi .NET

zastosowanie interfejsów API architektury WinRT w naturalny i prosty sposób, tak jakby to było czyste środowisko .NET. Z myślą o projektantach używających języka JavaScript opracowano rozszerzenie *WinJS* zapewniające dostęp do architektury WinRT, choć nadal muszą oni używać kodu HTML do budowania interfejsu użytkownika aplikacji.

Uwaga. Nawet pomimo tego, że kod HTML może być używany w aplikacjach Windows Apps, w dalszym ciągu ma on postać lokalnej aplikacji klienckiej, a nie aplikacji internetowej uzyskiwanej z serwera WWW.

Środowisko .NET Framework

Środowisko .NET Framework stanowi część systemu Windows. W tabeli 1.2 podano informacje o wersji tego środowiska zainstalowanego jako część danej wersji systemu Windows. Nowsza wersja środowiska może zostać jednak zainstalowana w nowszych wersjach systemu operacyjnego.

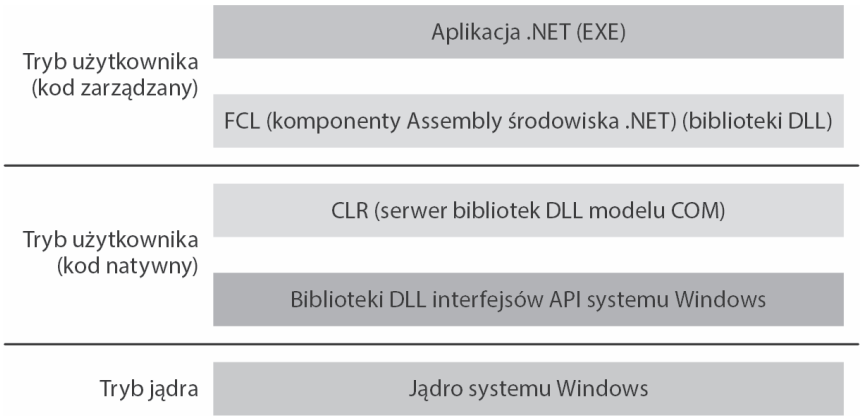
TABELA 1.2. *Domyślne instalacje środowiska .NET Framework w systemie Windows*

Wersja systemu Windows	Wersja środowiska .NET Framework
Windows 8	4.5
Windows 8.1	4.5.1
Windows 10	4.6
Windows 10 (wersja 1511)	4.6.1
Windows 10 (wersja 1607)	4.6.2

Środowisko .NET Framework złożone jest z następujących dwóch podstawowych komponentów:

- **Common Language Runtime (CLR)**. Jest to mechanizm uruchomieniowy środowiska .NET, który obejmuje kompilator Just In Time (JIT) dokonujący translacji instrukcji CIL (ang. *Common Intermediate Language*) na język maszynowy bazowego procesora sprzętowego, a także zarządcę „odśmiecania" pamięci, weryfikowanie typów, zabezpieczenia dostępu do kodu itp. Mechanizm implementowany jest jako serwer wewnątrzprocesowy modelu COM (biblioteka DLL), a ponadto korzysta z różnych rozwiązań zapewnianych przez interfejs API systemu Windows.
- **.NET Framework Class Library (FCL)**. Jest to duża kolekcja typów implementujących funkcjonalność, która jest zwykle wymagana przez aplikacje klienta i serwera, taką jak usługi interfejsu użytkownika, obsługa sieci, dostęp do bazy danych itp.

Oferując takie oraz inne elementy, w tym nowe języki programowania wysokiego poziomu (C#, Visual Basic, F#) oraz narzędzia wspierające, środowisko .NET Framework poprawia efektywność pracy projektanta, a ponadto zwiększa bezpieczeństwo i niezawodność w obrębie aplikacji opartych na tym środowisku. Na rysunku 1.1 pokazano relację między środowiskiem .NET Framework i systemem operacyjnym.



RYSUNEK 1.1. *Diagram prezentuje relację między środowiskiem .NET i systemem operacyjnym Windows*

Usługi, funkcje i programy

Kilka terminów w dokumentacji programisty i użytkownika systemu Windows ma odmienne znaczenie w różnych kontekstach. Na przykład termin *usługa* może odwoływać się do umożliwiającego wywołanie programu w systemie operacyjnym, sterownika urządzenia lub procesu serwerowego. Następująca lista pozwala się zorientować, jakie znaczenie w książce mają określone terminy:

- **Funkcje interfejsu API systemu Windows**. Są to udokumentowane podprogramy w interfejsie API systemu Windows umożliwiające wywołanie. Przykładowe funkcje obejmują `CreateProcess`, `CreateFile` i `GetMessage`.
- **Natywne usługi systemowe (lub wywołania systemowe)**. Są to nieudokumentowane usługi bazowe w systemie operacyjnym, które mogą być wywoływane w trybie użytkownika. Na przykład usługa `NtCreateUserProcess` to wewnętrzna usługa systemowa, jaką wywołuje funkcja `CreateProcess` systemu Windows w celu utworzenia nowego procesu.
- **Funkcje obsługi jądra (lub programy)**. Są to podprogramy wewnątrz systemu operacyjnego Windows, które mogą być wywoływane tylko w trybie jądra (zostanie zdefiniowany w dalszej części rozdziału). Na przykład `ExAllocatePoolWithTag` to program wywoływany przez sterowniki urządzeń w celu przydziału pamięci ze stert systemu Windows (nazywanych *pulami*).
- **Usługi systemu Windows**. Są to procesy uruchamiane przez menedżer sterowania usługami systemu Windows. Na przykład usługa Harmonogram zadań działa w obrębie procesu trybu użytkownika, który obsługuje polecenie `schtasks` (podobne do poleceń `at` i `cron` systemu UNIX). Zauważ, że choć w rejestrze sterowniki urządzeń systemu Windows są definiowane jako „usługi”, w książce nie są one postrzegane w ten sposób.
- **Biblioteki DLL (ang. *Dynamic Link Library*)**. Są to umożliwiające wywołanie podprogramy połączone razem jako plik binarny, który może być dynamicznie ładowany przez aplikacje korzystające z podprogramów. Przykłady takich bibliotek obejmują biblioteki *Msvcrt.dll* (biblioteka uruchomieniowa języka C) i *Kernel32.dll* (jedna z bibliotek podsystemu interfejsu API systemu Windows). Komponenty

i aplikacje trybu użytkownika systemu Windows intensywnie używają bibliotek DLL. W porównaniu z bibliotekami statycznymi biblioteki DLL oferują korzyść polegającą na tym, że aplikacje mogą je współużytkować, a system Windows zapewnia, że w przypadku aplikacji odwołujących się do kodu biblioteki DLL w pamięci znajduje się tylko jedna jego kopia. Zauważ, że komponenty Assembly środowiska .NET bibliotek są kompilowane jako biblioteki DLL, lecz bez żadnych niezarządzanych, wyeksportowanych podprogramów. Zamiast tego w celu uzyskania dostępu do odpowiednich typów i elementów członkowskich mechanizm CLR dokonuje analizy skompilowanych metadanych.

Procesy

Choć na pierwszy rzut oka programy i procesy wyglądają podobnie, fundamentalnie się różnią. *Program* to statyczna sekwencja instrukcji, natomiast *proces* to kontener zestawu zasobów używanych podczas wykonywania instancji programu. Na najwyższym poziomie abstrakcji proces systemu Windows składa się z następujących elementów:

- **Prywatny obszar adresów wirtualnych**. Jest to zestaw adresów wirtualnych pamięci, jakie mogą zostać użyte przez proces.
- **Program wykonywalny**. Termin ten definiuje początkowy kod i dane odwzorowywane na obszar adresów wirtualnych procesu.
- **Lista otwartych dojść**. Dojścia dokonują odwzorowania na różne zasoby systemowe, takie jak semafory, obiekty synchronizacji oraz pliki dostępne dla wszystkich wątków w procesie.
- **Kontekst zabezpieczeń**. Jest to *token dostępu* identyfikujący użytkownika, grupy zabezpieczeń, przywileje, atrybuty, oświadczenia, możliwości, stan wirtualizacji UAC (ang. *User Account Control*), sesję oraz stan ograniczonego konta użytkownika skojarzonego z procesem, a także identyfikator AppContainer i powiązane z nim informacje dotyczące „piaskownicy”.
- **Identyfikator procesu**. Unikatowy identyfikator, który wewnętrznie stanowi część identyfikatora nazywanego *identyfikatorem klienta*.
- **Co najmniej jeden wątek wykonywania**. Choć „pusty” proces jest możliwy, nie jest przydatny (przeważnie).

Dostępnych jest wiele narzędzi służących do wyświetlania (i modyfikowania) procesów oraz informacji o nich. W zamieszczonych dalej eksperymentach przedstawiono różne widoki informacji o procesach, jakie możesz uzyskać za pomocą niektórych z tych narzędzi. Choć wiele spośród tych narzędzi wchodzi w skład samego systemu Windows, a także jest obecnych w zestawach Debugging Tools for Windows i Windows SDK, część narzędzi jest osobno dostępnych w portalu Sysinternals. Wiele z tych narzędzi prezentuje pokrywające się podzbiory informacji o podstawowych procesach i wątkach. Podzbiory te są czasami identyfikowane przez różne nazwy.

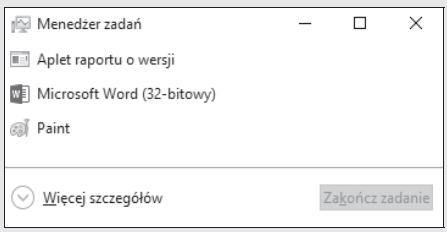
Menedżer zadań to prawdopodobnie najczęściej stosowane narzędzie do sprawdzania aktywności procesów (ponieważ w jądrze systemu Windows nie ma czegoś takiego jak „zadanie”, nazwa tego narzędzia jest trochę dziwna). W poniższym eksperymencie zaprezentowano niektóre z podstawowych funkcji Menedżera zadań.

EKSPERYMENT:
Wyświetlanie informacji o procesach za pomocą narzędzia Menedżer zadań

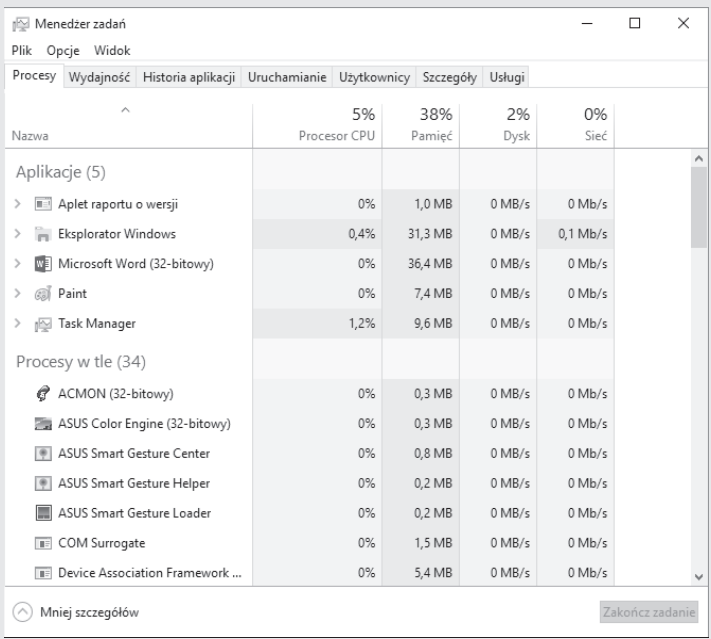
Wbudowane narzędzie Menedżer zadań Windows zapewnia uproszczoną listę procesów systemowych. Narzędzie możesz uruchomić na jeden z następujących czterech sposobów:

- Użycie kombinacji klawiszy *Ctrl+Shift+Esc*.
- Kliknięcie paska zadań prawym przyciskiem myszy, a następnie kliknięcie pozycji *Menedżer zadań*.
- Użycie kombinacji klawiszy *Ctrl+Alt+Delete* i kliknięcie przycisku *Menedżer zadań*.
- Uruchomienie pliku wykonywalnego *Taskmgr.exe*.

Przy pierwszym uruchomieniu Menedżer zadań znajduje się w trybie mniejszej liczby szczegółów, w przypadku którego pokazane są wyłącznie procesy z widocznym oknem najwyższego poziomu. Prezentuje to poniższy zrzut ekranu.



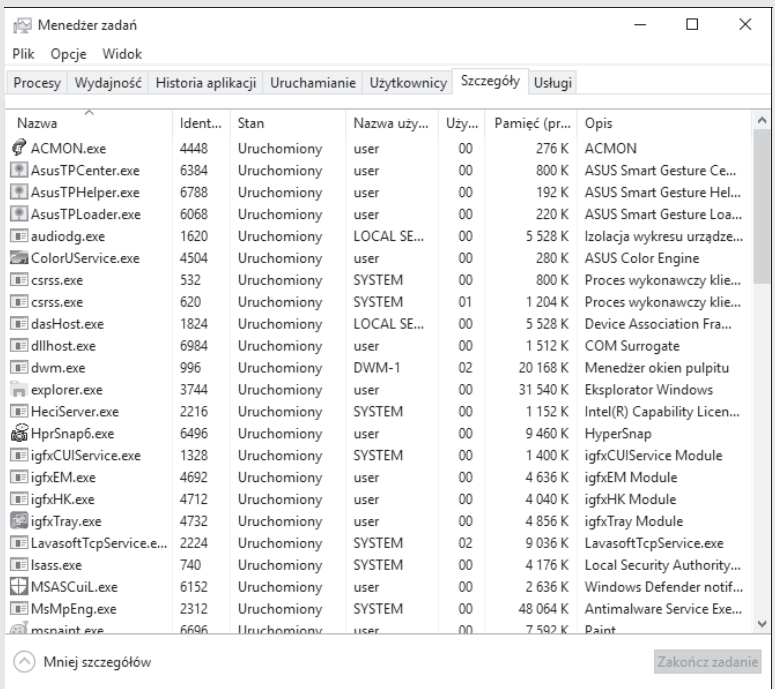
W wyświetlonym oknie niewiele możesz zdziałać, dlatego kliknij przycisk rozszerzający *Więcej szczegółów*, aby aktywować pełny widok Menedżera zadań. Karta *Procesy* powinna być domyślnie wybrana.



Karta *Procesy* zawiera listę procesów z następującymi czterema kolumnami: *Procesor CPU*, *Pamięć*, *Dysk* i *Sieć*. W celu wyświetlenia dodatkowych kolumn prawym przyciskiem myszy kliknij nagłówek widocznej kolumny. Dostępne kolumny to *Nazwa procesu*, *Identyfikator PID*, *Typ*, *Stan*, *Wydawca* i *Wiersz polecenia*. Niektóre procesy mogą być dodatkowo rozszerzone przez ukazanie widocznych okien najwyższego poziomu utworzonych przez proces.

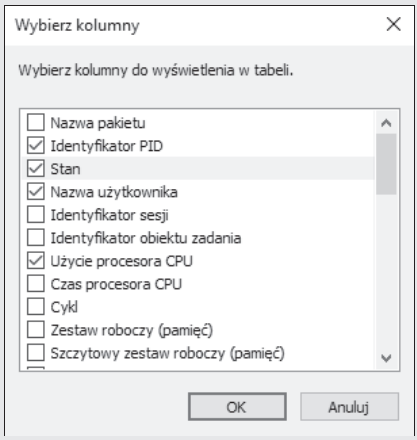
Aby uzyskać jeszcze więcej szczegółów dotyczących procesów, kliknij kartę *Szczegóły*. Możesz też prawym przyciskiem myszy kliknąć proces i wybrać pozycję *Przejdź do szczegółów* w celu aktywowania karty *Szczegóły* i wybrania konkretnego procesu.

Uwaga. Karta *Procesy* Menedżera zadań systemu Windows 7 w przybliżeniu odpowiada zawartości karty *Szczegóły* Menedżera zadań systemu Windows 8 lub nowszego. Karta *Aplikacje* Menedżera zadań systemu Windows 7 zawiera widoczne okna najwyższego poziomu, a nie procesy jako takie. W nowym Menedżerze zadań systemu Windows 8 lub nowszego informacje te są obecnie widoczne w obrębie karty *Procesy*.



Karta *Szczegóły* również prezentuje procesy, lecz w bardziej zwięzły sposób. Karta nie uwzględnia okien utworzonych przez procesy, a ponadto zapewnia więcej różnorodnych kolumn z informacjami.

Zauważ, że procesy są identyfikowane przez nazwę obrazu, którego są instancją. W przeciwieństwie do niektórych obiektów systemu Windows, procesom nie można nadać nazw globalnych. W celu wyświetlenia dodatkowych szczegółów prawym przyciskiem myszy kliknij wiersz nagłówków i kliknij pozycję *Wybierz kolumny*. Zostanie zaprezentowana lista kolumn.



Oto niektóre kluczowe kolumny:

- ***Wątki***. Kolumna zawiera liczbę wątków w każdym procesie. Liczba ta powinna standardowo wynosić co najmniej 1, ponieważ nie ma bezpośredniej możliwości utworzenia procesu bez żadnych wątków (poza tym taki proces jest naprawdę bezużyteczny). Jeśli proces nie zawiera żadnych wątków, oznacza to zwykle, że z jakiegoś powodu proces nie może zostać usunięty (prawdopodobnie na skutek jakiegoś kodu sterownika z błędami).
- ***Dojścia***. Kolumna prezentuje liczbę dojść obiektów jądra otwartych przez wątki działające w obrębie procesu (zostanie to opisane w dalszej części niniejszego rozdziału).
- ***Stan***. Kolumna ta jest trochę zawiła. W wypadku procesów pozbawionych jakiegokolwiek interfejsu użytkownika normalną sytuacją powinien być stan *Uruchomiony*, choć wszystkie wątki mogą na coś oczekiwać (np. przesyłanie sygnału do obiektu jądra lub zakończenie operacji wejścia-wyjścia). W przypadku takich procesów inna opcja to stan *Wstrzymany*, który ma miejsce, gdy wszystkie wątki procesu znajdują się w stanie wstrzymania. Spowodowanie czegoś takiego przez sam proces jest mało prawdopodobne, ale może zostać osiągnięte w sposób programowy przez wywołanie dla procesu nieudokumentowanego, natywnego interfejsu API `NtSuspendProcess`, co zwykle odbywa się za pośrednictwem narzędzia (na przykład odpowiednią opcję oferuje opisane dalej narzędzie Process Explorer). W wypadku procesów tworzących interfejs użytkownika stan *Uruchomiony* oznacza, że interfejs ten jest aktywny. Inaczej mówiąc, wątek, który utworzył okno lub okna, oczekuje na dane wejściowe interfejsu użytkownika (z technicznego punktu widzenia jest to kolejka komunikatów powiązana z wątkiem). Stan *Wstrzymany* jest możliwy tak jak w przypadku braku interfejsu, ale dla aplikacji Windows Apps (udostępniających architekturę Windows Runtime) stan ten występuje normalnie, gdy w wyniku zminimalizowania przez użytkownika aplikacja utraci swój status pierwszoplanowej. Takie procesy są wstrzymywane po pięciu sekundach, dlatego nie zużywają żadnych zasobów procesora lub zasobów sieciowych. Dzięki temu nowa aplikacja pierwszoplanowa ma możliwość skorzystania z wszystkich zasobów komputera. Jest to szczególnie ważne w przypadku urządzeń zasilanych z baterii, takich jak tablety i telefony. Trzecia możliwa wartość w kolumnie *Stan* to stan *Nie odpowiada*.

Może on wystąpić, gdy wątek w procesie, który utworzył interfejs użytkownika, nie sprawdził przez co najmniej 5 sekund swojej kolejki komunikatów pod kątem aktywności związanej z interfejsem. Proces (a faktycznie wątek będący w posiadaniu okna) może być zajęty realizowaniem jakiegoś zadania intensywnie obciążającego procesor lub oczekiwać na zupełnie coś innego (np. zakończenie operacji wejścia-wyjścia). W każdym razie interfejs użytkownika jest zablokowany, a system Windows wskazuje to przez przyciemnienie odpowiednich okien i dołączenie do ich paska tytułu komunikatu *(Nie odpowiada)*.

Każdy proces wskazuje również na swój proces nadrzędny lub proces tworzący (który nie zawsze w wypadku tego procesu jest procesem tworzącym). Jeśli proces nadrzędny już nie istnieje, odpowiednia informacja nie jest aktualizowana. Możliwe jest zatem, że proces odwołuje się do nieistniejącego procesu nadrzędnego. Nie stanowi to problemu, ponieważ nic nie jest zależne od zapewnienia aktualności tej informacji. W przypadku narzędzia Process Explorer pod uwagę brany jest czas uruchomienia procesu nadrzędnego, co ma na celu uniknięcie dołączania procesu podrzędnego na podstawie zastosowanego identyfikatora procesu. Działanie to ilustruje poniższy eksperyment.

Uwaga. Dlaczego proces nadrzędny nie będzie taki sam jak jego proces tworzący? W określonych sytuacjach niektóre procesy, które wydają się tworzone przez konkretną aplikację użytkownika, mogą korzystać z pomocy brokera lub procesu pomocniczego odpowiedzialnego za wywołanie interfejsu API tworzenia procesów. W takich przypadkach wprowadzające w błąd (a czasem niepoprawne, jeśli niezbędne jest dziedziczenie obszaru adresów lub dojścia) byłoby wyświetlenie procesu brokera jako procesu tworzącego. W związku z tym ma miejsce operacja nazywana ponownym określaniem procesu nadrzędnego. Odpowiedni przykład zaprezentowano w rozdziale 7., „Bezpieczeństwo”.

EKSPERYMENT: Wyświetlanie drzewa procesów

Identyfikator procesu nadrzędnego lub procesu tworzącego to unikatowy atrybut związany z procesem, który nie jest wyświetlany przez większość narzędzi. Wartość tego atrybutu możesz uzyskać za pomocą narzędzia Monitor wydajności (lub w sposób programowy), tworząc żądanie dotyczące identyfikatora procesu tworzącego. Narzędzie *Tlist.exe* należące do zestawu Debugging Tools for Windows umożliwia wyświetlenie drzewa procesów przy użyciu opcji `/t`. Oto przykładowe dane wynikowe polecenia `tlist /t`:

```
System Process (0)
System (4)
  smss.exe (360)
csrss.exe (460)
wininit.exe (524)
  services.exe (648)
    svchost.exe (736)
      unsecapp.exe (2516)
      WmiPrvSE.exe (2860)
      WmiPrvSE.exe (2512)
      RuntimeBroker.exe (3104)
      SkypeHost.exe (2776)
      ShellExperienceHost.exe (3760) Host środowiska powłoki systemu Windows
```

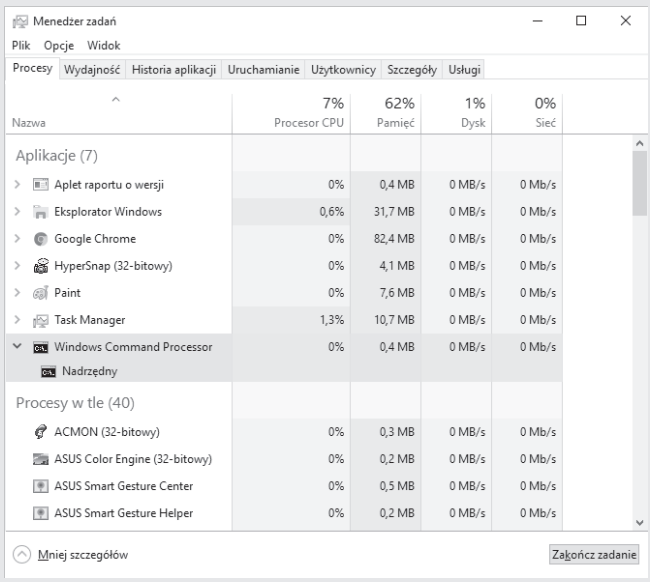
```
ApplicationFrameHost.exe (2848) OleMainThreadWndName
      SearchUI.exe (3504) Wyszukiwarka
      WmiPrvSE.exe (1576)
      TiWorker.exe (6032)
      wuapihost.exe (5088)
    svchost.exe (788)
    svchost.exe (932)
    svchost.exe (960)
    svchost.exe (976)
    svchost.exe (68)
    svchost.exe (380)
    VSSVC.exe (1124)
    svchost.exe (1176)
      sihost.exe (3664)
      taskhostw.exe (3032) Task Host Window
    svchost.exe (1212)
    svchost.exe (1636)
    spoolsv.exe (1644)
    svchost.exe (1936)
    OfficeClickToRun.exe (1324)
    MSOIDSVC.EXE (1256)
      MSOIDSVCM.EXE (2264)
    MBAMAgent.exe (2072)
    MsMpEng.exe (2116)
    SearchIndexer.exe (1000)
      SearchProtocolHost.exe (824)
    svchost.exe (3328)
    svchost.exe (3428)
    svchost.exe (4400)
    svchost.exe (4360)
    svchost.exe (3720)
    TrustedInstaller.exe (6052)
  lsass.exe (664)
csrss.exe (536)
winlogon.exe (600)
  dwm.exe (1100) DWM Notification Window
explorer.exe (3148) Program Manager
  OneDrive.exe (4448)
  cmd.exe (5992) C:\windows\system32\cmd.exe - tlist  /t
    conhost.exe (3120) CicMarshalWnd
    tlist.exe (5888)
SystemSettingsAdminFlows.exe (4608)
```

Na liście zastosowano wcięcie dla każdego procesu, aby pokazać jego relację między procesem nadrzędnym i podrzędnym. Procesy, których procesy nadrzędne nie istnieją, są wyrównywane do lewej strony (tak jak proces `explorer.exe` w powyższym przykładzie), ponieważ jeśli nawet występuje proces nadrzędny względem procesu nadrzędnego danego procesu, nie ma możliwości znalezienia tej relacji. System Windows utrzymuje jedynie identyfikator procesu tworzącego dany proces, a nie odnośnik do procesu nadrzędnego względem tego procesu tworzącego itd.

Liczba podana w nawiasach okrągłych to identyfikator procesu, a tekst następujący po nim w przypadku niektórych procesów to tytuł okna utworzonego przez dany proces.

Aby potwierdzić, że system Windows nie śledzi więcej informacji niż tylko identyfikator procesu nadrzędnego, wykonaj następujące kroki:

1. Użyj kombinacji klawisza z logo systemu Windows i klawisza *R*, wpisz polecenie **`cmd`** i naciśnij klawisz *Enter*, aby otworzyć okno wiersza poleceń.
2. Wpisz polecenie **`title Nadrzędny`** w celu zmiany tytułu okna na *Nadrzędny*.
3. Wpisz polecenie **`start cmd`**, aby otworzyć drugie okno wiersza poleceń.
4. W oknie tym wpisz polecenie **`title Podrzędny`**.
5. W tym samym oknie wpisz polecenie **`mspaint`**, aby uruchomić program Microsoft Paint.
6. Powróć do drugiego okna wiersza poleceń i wpisz polecenie **`exit`**. Zauważ, że program Paint pozostaje aktywny.
7. Użyj kombinacji klawiszy *Ctrl+Shift+Esc* w celu otwarcia Menedżera zadań.
8. Jeśli Menedżer zadań działa w trybie mniejszej liczby szczegółów, kliknij przycisk *Więcej szczegółów*.
9. Kliknij kartę *Procesy*.
10. Znajdź aplikację *Windows Command Processor* i rozwiń jej węzeł. Powinien być widoczny tytuł *Nadrzędny*, tak jak na poniższym zrzucie ekranu.



11. Prawym przyciskiem myszy kliknij wpis *Windows Command Processor* i wybierz pozycję *Przejdź do szczegółów*.
12. Prawym przyciskiem myszy kliknij proces *cmd.exe* i wybierz pozycję *Zakończ drzewo procesów*.
13. W oknie dialogowym potwierdzenia Menedżera zadań kliknij pozycję *Zakończ drzewo procesów*.

Pierwsze okno wiersza poleceń zniknie, ale w dalszym ciągu powinno być widoczne okno programu Paint, ponieważ było ono „wnukiem” zakończonego procesu wiersza poleceń. Ze względu na to, że zakończono proces pośredni (nadrzędny względem programu Paint), nie ma łącza między jego procesem nadrzędnym („dziadkiem”) i „wnukiem”.

Narzędzie Process Explorer dostępne w witrynie Sysinternals zapewnia więcej szczegółów dotyczących procesów i wątków niż jakiekolwiek inne oferowane narzędzie. Z tego właśnie powodu zauważysz, że zostało ono użyte w kilku eksperymentach zamieszczonych w książce. Oto niektóre z unikatowych cech prezentowanych lub zapewnianych przez narzędzie Process Explorer:

- Token zabezpieczeń procesu, taki jak listy grup i przywilejów oraz stan wirtualizacji.
- Wyróżnianie w celu ukazania zmian w procesie, wątku, bibliotekach DLL i liście dojść.
- Lista usług wewnątrz procesów udostępniających usługi, w tym nazwa wyświetlana i opis.
- Lista dodatkowych atrybutów procesu, takich jak zasady minimalizowania oraz ich ochrona procesu.
- Procesy stanowiące część zadania oraz jego szczegóły.
- Procesy udostępniające aplikacje .NET i szczegóły dotyczące środowiska .NET, takie jak lista elementów AppDomain, ładowane komponenty Assembly oraz liczniki wydajności CLR.
- Procesy udostępniające architekturę Windows Runtime (procesy imersywne).
- Czas uruchomienia procesów i wątków.
- Kompletna lista plików mapowanych w pamięci (nie tylko biblioteki DLL).
- Możliwość wstrzymywania procesu lub wątku.
- Możliwość kończenia działania poszczególnych wątków.
- Łatwe identyfikowanie tego, jakie procesy wykorzystały w danym czasie najwięcej zasobów procesora.

Uwaga. Monitor wydajności może informować o wykorzystaniu procesora przez procesy dla danego ich zestawu, ale nie wyświetli automatycznie procesów utworzonych po rozpoczęciu sesji monitorowania wydajności. Umożliwia to jedynie ręczne śledzenie z wykorzystaniem binarnego formatu danych wyjściowych.

Narzędzie Process Explorer zapewnia również w jednym miejscu łatwy dostęp do następujących informacji:

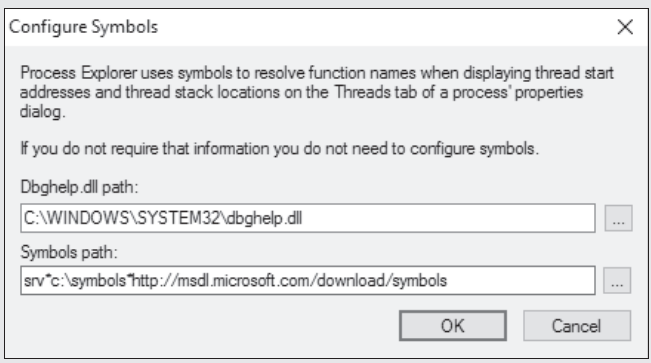
- Drzewo procesów pozwalające na zwijanie jego części.
- Dojścia otwarte w procesie, w tym dojścia bez nazwy.
- Lista bibliotek DLL (i plików mapowanych w pamięci) w procesie.
- Aktywność wątków w procesie.
- Stosy wątków trybu użytkownika i trybu jądra, w tym mapowanie adresów na nazwy za pomocą biblioteki *Dbghelp.dll* wchodzącej w skład zestawu Debugging Tools for Windows.
    - Dokładniejsza wartość procentowa związana z wykorzystaniem procesora, która bazuje na liczbie cykli wątków. Jest to jeszcze lepsza reprezentacja dokładnej aktywności procesora, co zostało objaśnione w rozdziale 4., „Wątki”.
    - Poziom integralności.
- Szczegóły dotyczące menedżera pamięci, takie jak maksymalna ilość pamięci zadeklarowanej oraz limity stronicowanej pamięci jądra i puli niestronicowanej (inne narzędzia podają jedynie bieżącą wielkość).

Poniżej zamieszczono wprowadzający eksperyment wykorzystujący narzędzie Process Explorer.

EKSPERYMENT:
Wyświetlanie szczegółów procesu za pomocą narzędzia Process Explorer

Pobierz najnowszą wersję narzędzia Process Explorer z witryny Sysinternals i uruchom je z uprawnieniami standardowego użytkownika. Alternatywnie prawym przyciskiem myszy kliknij plik wykonywalny i z menu wybierz polecenie *Uruchom jako administrator*, aby narzędzie uruchomić z uprawnieniami administratora. Spowoduje to zainstalowanie przez narzędzie Process Explorer sterownika oferującego więcej funkcji. Poniższy opis dotyczy dowolnego sposobu uruchomienia tego narzędzia.

Przy pierwszym załadowaniu narzędzia należy skonfigurować symbole. W przeciwnym razie po dwukrotnym kliknięciu procesu i kliknięciu karty *Threads* pojawi się komunikat informujący o tym, że aktualnie nie są skonfigurowane symbole. W przypadku poprawnego skonfigurowania symboli narzędzie Process Explorer może uzyskać dostęp do informacji o symbolach w celu wyświetlenia nazwy symbolicznej funkcji uruchamiającej wątek, a także funkcji znajdujących się na stosie wywołania wątku. Jest to przydatne przy identyfikowaniu tego, jakie działania wątki realizują wewnątrz procesu. W celu skorzystania z symboli musi zostać zainstalowany zestaw Debugging Tools for Windows (zostanie to opisane w dalszej części rozdziału). Kliknij następnie menu *Options*, wybierz pozycję *Configure Symbols* oraz podaj ścieżkę do pliku *Dbghelp.dll* w folderze zestawu Debugging Tools i poprawną ścieżkę do symboli. Na przykład w systemie 64-bitowym taka konfiguracja będzie poprawna, jeśli zestaw Debugging Tools for Windows zainstalowano w położeniu domyślnym jako część zestawu WDK.

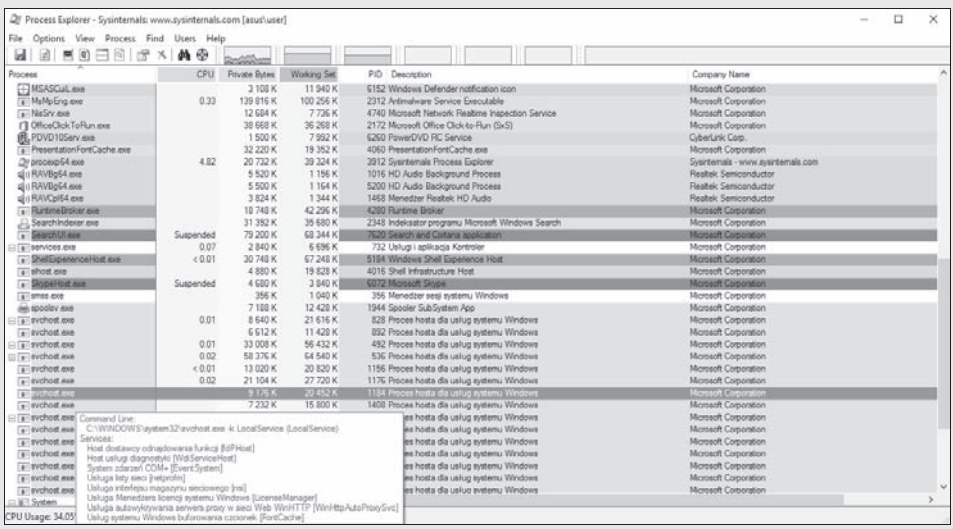


W powyższym przykładzie użyto serwera symboli na żądanie, aby zapewnić dostęp do symboli. Kopia plików symboli przechowywana jest na komputerze lokalnym w folderze *C:\symbols* (jeśli wolne miejsce na dysku stanowi problem, folder może zostać zastąpiony innym folderem, takim jak zlokalizowany na innym dysku). Więcej informacji o konfigurowaniu serwera symboli znajdziesz pod adresem *https://msdn.microsoft.com/en-us/library/windows/desktop/ee416588.aspx*.

Wskazówka. Serwer symboli firmy Microsoft możesz skonfigurować przez ustawienie dla zmiennej środowiskowej o nazwie `_NT_SYMBOL_PATH` wartości podanej na zamieszczonym wcześniej zrzucie ekranu. Różne narzędzia, takie między innymi jak Process Explorer, debuggery wchodzące w skład zestawu Debugging Tools for Windows i Visual Studio, szukają automatycznie tej wartości. Ułatwi to uniknięcie konieczności osobnego konfigurowania każdego narzędzia.

Po uruchomieniu narzędzie Process Explorer wyświetla domyślnie widok drzewa procesów. Możesz rozwinąć dolny panel, aby ujrzeć otwarte dojścia lub mapowane biblioteki DLL i pliki mapowane w pamięci. Narzędzie prezentuje też podpowiedź dotyczącą ścieżki i wiersza poleceń procesu, która staje się widoczna w momencie umieszczenia kursora myszy nad nazwą procesu. W wypadku niektórych typów procesów podpowiedź zawiera również dodatkowe informacje. Oto one:

- Usługi w obrębie procesu udostępniającego usługi (np. *Svchost.exe*).
- Zadania w obrębie procesu udostępniającego zadania (np. *TaskHostw.exe*).
- Element docelowy procesu *Rundll32.exe*, który używany jest na potrzeby elementów panelu sterowania i innych funkcji.
- Informacje o klasie modelu COM podczas udostępniania wewnątrz procesu *Dllhost.exe* (określany też mianem domyślnego komponentu COM+ Surrogate).
- Informacje o dostawcy na potrzeby procesów hosta WMI (*Windows Management Instrumentation*), takich jak *WMIPrvSE.exe*.
- Informacje dotyczące procesów aplikacji Windows Apps (procesy udostępniające architekturę Windows Runtime, które w skrócie opisano wcześniej w rozdziale w podpunkcie „Architektura Windows Runtime”).



Oto kroki pozwalające na zaznajomienie się z podstawowymi możliwościami narzędzia Process Explorer:

1. Zauważ, że procesy udostępniające usługi są domyślnie wyróżnione na różowo. Procesy utworzone samodzielnie są wyróżnione na niebiesko. Aby zmienić te kolory, wyświetl menu rozwijane, a następnie wybierz pozycje *Options* i *Configure Colors*.
2. Kursor myszy umieść nad nazwą obrazu procesów. Zauważ, że podpowiedź wyświetla pełną ścieżkę. Jak już wspomniano, w wypadku niektórych typów procesów w podpowiedzi widoczne są dodatkowe szczegóły.
3. Po kliknięciu pozycji *View* i *Select Columns* na karcie *Process Image* zaznacz opcję *Image Path*.
4. Aby posortować procesy, kliknij nagłówek kolumny *Process*. Zauważ, że znika widok drzewa (możesz wyświetlić ten widok lub dokonać sortowania według dowolnej z widocznych kolumn). Ponownie kliknij nagłówek kolumny *Process*, aby zastosować sortowanie w kolejności od Z do A. Kliknij nagłówek trzeci raz w celu przywrócenia widoku drzewa.
5. Wyświetl menu *View* i wyłącz zaznaczenie opcji *Show Processes from All Users*, aby zostały wyświetlone tylko procesy utworzone samodzielnie.
6. Kliknij menu *Options*, wybierz pozycję *Difference Highlight Duration* i zmień wartość na 3 sekundy. Wywołaj następnie nowy proces (jakikolwiek). Zauważ, że przez 3 sekundy nowy proces wyróżniony jest na zielono. Zakończ ten nowy proces. Zwróć uwagę na to, że zanim proces przestanie być widoczny, przez 3 sekundy jest on wyróżniony na czerwono. Może to być przydatne przy obserwacji procesów tworzonych i kończonych w systemie.
7. Dwukrotnie kliknij proces i w obrębie wyświetlonych właściwości procesu sprawdź różne dostępne karty (karty będą używane w różnych eksperymentach zamieszczonych w książce, w których objaśniono prezentowane informacje).

Wątki

Wątek to obiekt wewnątrz procesu, którego wykonanie planuje system Windows. Bez wątku niemożliwe byłoby uruchomienie programu procesu. Wątek uwzględnia następujące zasadnicze komponenty:

- Zawartość zestawu rejestrów procesora reprezentujących jego stan.
- Dwa stosy — jeden dla wątku używany podczas wykonywania w trybie jądra oraz jeden na potrzeby wykonywania w trybie użytkownika.
- Prywatny obszar pamięci nazywany *pamięcią lokalną wątku TLS* (ang. *Thread Local Storage*) przeznaczony dla podsystemów, bibliotek uruchomieniowych i bibliotek DLL.
- Unikatowy identyfikator nazywany *identyfikatorem wątku* (część wewnętrznej struktury określanej mianem *identyfikatora klienta*; identyfikatory procesów i wątków są generowane poza tą samą przestrzenią nazw, dlatego nigdy się nie nakładają).

Ponadto wątki mają czasami własny kontekst zabezpieczeń lub token, który często używany jest przez wielowątkowe aplikacje serwerowe naśladujące kontekst zabezpieczeń obsługiwanych przez nie klientów.

Zmienne rejestry, stosy i prywatny obszar pamięci są określane mianem *kontekstu* wątku. Ponieważ związane z tym informacje są inne dla każdej architektury sprzętowej z uruchomionym systemem Windows, z konieczności taka struktura jest powiązana z architekturą. Funkcja `GetThreadContext` systemu Windows zapewnia dostęp do informacji specyficznych dla danej architektury (nazywanych blokiem `CONTEXT`).

Ponieważ przełączanie wykonywania między dwoma wątkami angażuje dyspozytora jądra, może to być wymagająca operacja, zwłaszcza wtedy, gdy dwa wątki często wzajemnie się przełączają. Aby zmniejszyć związany z tym koszt, system Windows implementuje dwa mechanizmy: *włókna* (ang. *fibers*) i *planowanie w trybie użytkownika UMS* (ang. *User Mode Scheduling*).

Uwaga. Wątki aplikacji 32-bitowej działającej w 64-bitowym systemie Windows będą zawierać zarówno 32-bitowe, jak i 64-bitowe konteksty, których podsystem Wow64 (Windows on Windows) użyje w razie potrzeby do przełączania uruchomionej aplikacji z trybu 32-bitowego do trybu 64-bitowego. Wątki te będą mieć dwa stosy użytkownika oraz dwa bloki `CONTEXT`, a zwykłe funkcje API systemu Windows będą zwracać kontekst 64-bitowy. Funkcja `Wow64GetThreadContext` zwróci jednak kontekst 32-bitowy.

Włókna

Włókna umożliwiają aplikacji planowanie jej własnych wątków wykonania, zamiast polegać na wbudowanym w system Windows mechanizmie planowania opartym na priorytetach. Włókna są często nazywane *wątkami lekkimi*. Z punktu widzenia planowania włókna są niewidoczne dla jądra, ponieważ implementowane są w trybie użytkownika w bibliotece *Kernel32.dll*. Aby zastosować włókna, tworzone jest najpierw wywołanie funkcji `ConvertThreadToFiber` systemu Windows. Przekształca ona wątek w działające włókno. Dalej nowe włókno uzyskane w wyniku przekształcenia może tworzyć dodatkowe włókna za pośrednictwem funkcji `CreateFiber` (każde włókno może mieć własny zestaw włókien). W przeciwieństwie jednak do wątku, włókno nie rozpoczyna wykonywania do momentu ręcznego wybrania go za pomocą wywołania funkcji `SwitchToFiber`. Nowe włókno jest uruchomione do chwili jego zakończenia lub wywołania przez nie tej funkcji, co ponownie powoduje wybranie kolejnego włókna do uruchomienia. Więcej informacji na temat funkcji włókien zamieszczono w dokumentacji zestawu SDK systemu Windows.

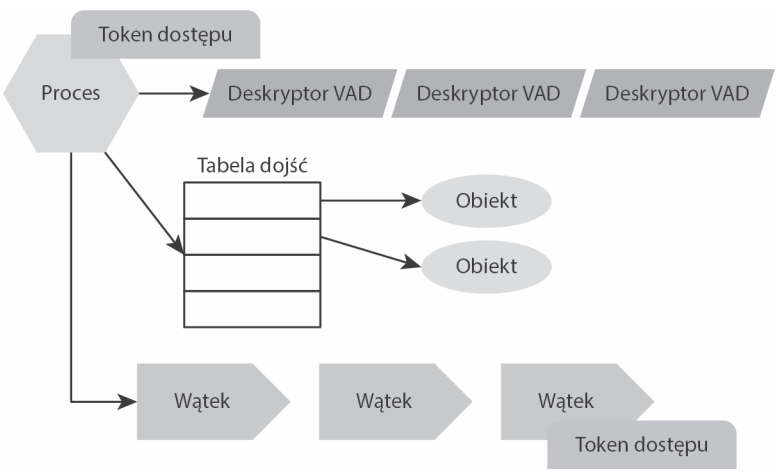
Uwaga. Użycie włókien nie jest zwykle dobrym pomysłem. Wynika to stąd, że są one niewidoczne dla jądra. Powodują one ponadto takie problemy, jak współużytkowanie pamięci lokalnej wątku (TLS), ponieważ kilka włókien może działać w tym samym wątku. Choć istnieje pamięć lokalna włókna FLS (ang. *Fiber Local Storage*), nie rozwiązuje ona wszystkich problemów ze współużytkowaniem, a niezależnie od tego włókna związane z operacjami wejścia-wyjścia będą kiepsko działać. Ponadto włókna nie mogą działać jednocześnie na więcej niż jednym procesorze i są ograniczone wyłącznie do wielozadaniowości równoległej. W większości sytuacji najlepszym rozwiązaniem jest umożliwienie jądru systemu Windows obsługi planowania za pomocą odpowiednich wątków dostępnego zadania.

Wątki planowania w trybie użytkownika

Wątki planowania w trybie użytkownika (UMS), które są dostępne wyłącznie w 64-bitowych wersjach systemu Windows, zapewniają te same podstawowe korzyści co włókna, a ponadto mają tylko kilka mankamentów. Wątki UMS mają własny stan wątków jądra, dlatego są widoczne dla jądra, co pozwala wielu wątkom UMS tworzyć blokujące wywołania systemowe, a także współużytkować zasoby i rywalizować o nie. Gdy dwa lub więcej wątków UMS wymaga zrealizowania działań w trybie użytkownika, mogą w nim okresowo przełączać konteksty wykonywania (przez przechodzenie z jednego wątku do drugiego), a nie angażować do tego dyspozytora. Z perspektywy jądra w dalszym ciągu działa ten sam jego wątek i nic nie uległo zmianie. Gdy wątek UMS wykonuje operację wymagającą uzyskania dostępu do jądra (na przykład w przypadku wywołania systemowego), dokonuje przełączenia do własnego dedykowanego wątku trybu jądra (jest to określane mianem *bezpośredniego przełączenia kontekstu*). Choć współbieżne wątki UMS w dalszym ciągu nie mogą działać na wielu procesorach, są one zgodne z modelem wywłaszczalnym, który nie do końca oferuje możliwości współpracy.

Chociaż wątki mają własny kontekst wykonywania, każdy wątek w procesie współużytkuje jego obszar adresów wirtualnych (oprócz reszty zasobów należących do procesu). Oznacza to, że wszystkie wątki w procesie dysponują pełnym (z odczytem i zapisem) dostępem do obszaru adresów wirtualnych procesu. Wątki nie mogą jednak przypadkowo odwoływać się do obszaru adresów innego procesu, chyba że inny proces udostępnia część swojego prywatnego obszaru adresów jako sekcję wspólnej pamięci (w interfejsie API systemu Windows nazywanej *obiektem mapowania plików*) lub jeden proces ma możliwość otwarcia innego procesu w celu użycia międzyprocesowych funkcji pamięci, takich jak `ReadProcessMemory` i `WriteProcessMemory` (z których domyślnie może skorzystać proces działający na bazie tego samego konta użytkownika, a nie wewnątrz komponentu AppContainer lub innego typu „piaskownicy”, chyba że dla procesu docelowego zastosowano określone elementy ochrony).

Oprócz prywatnego obszaru adresów i jednego lub większej liczby wątków, każdy proces ma kontekst zabezpieczeń oraz listę otwartych dojść powiązanych z obiektami jądra, takimi jak pliki i sekcje wspólnej pamięci, lub z jednym z obiektów synchronizacji, takich jak obiekty *mutex*, zdarzenia lub semafory. Zilustrowano to na rysunku 1.2.



RYSUNEK 1.2. *Proces i jego zasoby*

Kontekst zabezpieczeń każdego procesu jest przechowywany w obiekcie nazywanym *tokenem dostępu*. Token dostępu procesu zawiera związaną z nim identyfikację zabezpieczeń i dane uwierzytelniające. Domyślnie wątki nie mają własnego tokenu dostępu, ale mogą go uzyskać, dzięki czemu poszczególne wątki mają możliwość naśladowania kontekstu zabezpieczeń innego procesu (w tym procesów zdalnego systemu Windows) bez wpływu na inne wątki procesu (w rozdziale 7. zamieszczono więcej szczegółów dotyczących zabezpieczeń procesów i wątków).

Deskryptory adresów wirtualnych VAD (*Virtual Address Descriptor*) to struktury danych używane przez menedżer pamięci do śledzenia adresów wirtualnych wykorzystywanych przez proces. Te struktury danych opisano obszerniej w rozdziale 5.

Zadania

System Windows zapewnia rozszerzenie modelu procesów nazywane *zadaniem*. Główną funkcją obiektu zadania jest umożliwienie przetwarzania grup procesów jako jednostki oraz zarządzania nimi. Obiekt zadania pozwala kontrolować określone atrybuty oraz zapewnia limity procesowi lub procesom powiązanym z zadaniem. Obiekt ten zapisuje też podstawowe informacje rejestracyjne dla wszystkich procesów powiązanych z zadaniem, a także dla procesów, które zostały z nim skojarzone, ale później zostały zakończone. Pod pewnymi względami obiekt zadania kompensuje brak w systemie Windows strukturalnego drzewa procesów, a przy tym w wielu aspektach oferuje większe możliwości niż drzewo procesów w stylu obecnego w systemie UNIX.

Uwaga. Narzędzie Process Explorer może prezentować procesy zarządzane przez zadanie z wykorzystaniem domyślnego koloru brązowego, co jednak standardowo nie jest umożliwione (w celu aktywowania tej funkcji wyświetl menu *Options* i wybierz pozycję *Configure Colors*). Ponadto strony właściwości takiego procesu oferują dodatkową kartę *Job*, która zapewnia informacje o samym obiekcie zadania.

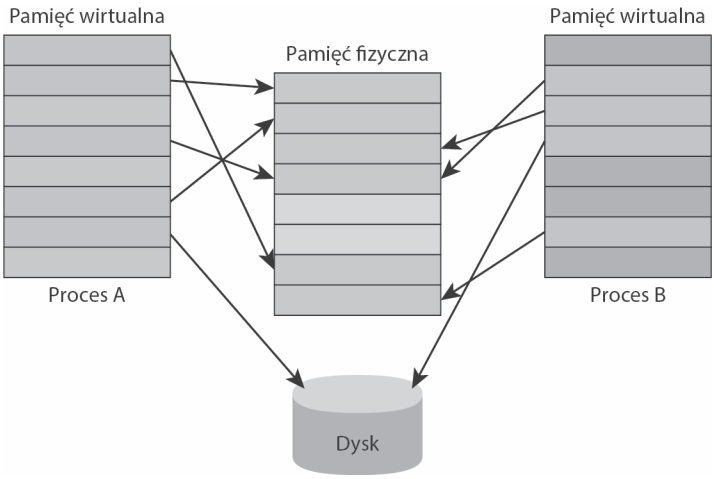
Znacznie więcej na temat wewnętrznej struktury procesów i zadań znajdziesz w rozdziale 3. W rozdziale 4. przeczytasz o wątkach i ich algorytmach planowania.

Pamięć wirtualna

System Windows implementuje system pamięci wirtualnej oparty na płaskim (liniowym) obszarze adresów, który zapewnia każdemu procesowi „wrażenie" dysponowania własnym dużym, prywatnym obszarem adresów. Pamięć wirtualna oferuje widok logiczny pamięci, który może nie odpowiadać jej układowi fizycznemu. W środowisku uruchomieniowym menedżer pamięci wspierany sprzętowo dokonuje translacji lub *mapowania* adresów wirtualnych na adresy fizyczne, w których dane są faktycznie przechowywane. Kontrolując elementy ochrony i mapowanie, system operacyjny może zapewnić, że poszczególne procesy nie kolidują ze sobą lub nadpisują dane systemowe.

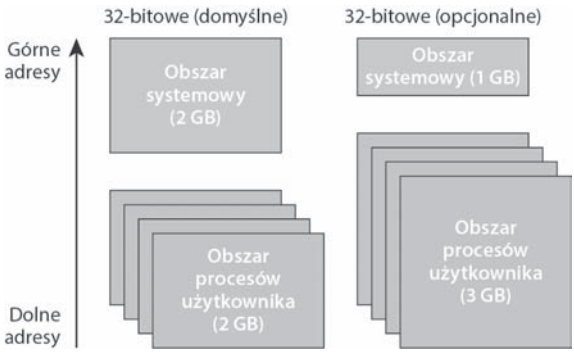
Ponieważ większość systemów dysponuje znacznie mniejszą ilością pamięci fizycznej, niż wynosi wielkość pamięci wirtualnej używanej przez działające procesy, menedżer pamięci transferuje lub *stronicuje* na dysk część zawartości pamięci. Stronicowanie danych na dysku powoduje zwolnienie pamięci fizycznej, dzięki czemu może ona zostać przeznaczona dla innych procesów lub samego systemu operacyjnego. Gdy wątek uzyskuje dostęp do adresu wirtualnego, dla którego wykonano operację stronicowania na dysk, menedżer pamięci wirtualnej ładuje informacje ponownie do pamięci z dysku.

Aplikacje nie muszą być modyfikowane w żaden sposób, aby mogły skorzystać ze stronicowania, ponieważ obsługa sprzętowa umożliwia menedżerowi pamięci przeprowadzanie stronicowania bez wiedzy lub udziału procesów lub wątków. Na rysunku 1.3 pokazano dwa procesy korzystające z pamięci wirtualnej, w której część fragmentów jest mapowana na pamięć fizyczną (RAM), a część jest stronicowana na dysk. Zauważ, że ciągłe porcje pamięci wirtualnej mogą być mapowane na nieciągłe porcje w pamięci fizycznej. Porcje takie są nazywane stronami i mają domyślną wielkość wynoszącą 4 kB.



RYSUNEK 1.3. *Mapowanie pamięci wirtualnej na pamięć fizyczną z zastosowaniem stronicowania*

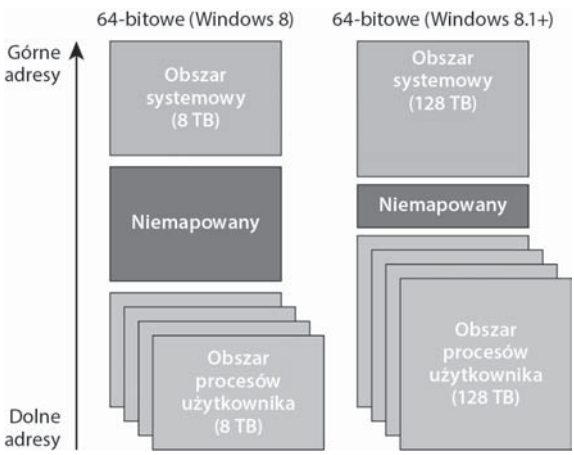
Wielkość obszaru adresów wirtualnych jest inna dla każdej platformy sprzętowej. W wypadku 32-bitowych systemów x86 całkowity obszar adresów wirtualnych ma teoretyczną, maksymalną wielkość równą 4 GB. Domyślnie system Windows alokuje dolną połowę tego obszaru adresów (adresy od 0x00000000 do 0x7FFFFFFF) procesom na potrzeby ich unikatowej pamięci prywatnej, a górną połowę (adresy od 0x80000000 do 0xFFFFFFFF) przeznacza do wykorzystania własnej pamięci chronionej. Mapowania dolnej połowy obszaru zmieniają się, aby uwzględnić obszar adresów wirtualnych aktualnie wykonywanego procesu. Z kolei większość mapowań górnej połowy zawsze jest złożona z pamięci wirtualnej systemu operacyjnego. System Windows oferuje opcje rozruchowe, takie jak kwalifikator `increaseuserva` w bazie danych Boot Configuration Database (opisano ją w rozdziale 5.), które zapewniają procesom uruchamiającym specjalnie oznaczone programy możliwość użycia maksymalnie 3 GB prywatnego obszaru adresów, pozostawiając systemowi 1 GB (pod terminem „specjalnie oznaczone" kryje się to, że w nagłówku obrazu wykonywalnego musi zostać ustawiona flaga dużego obszaru adresów). Taka opcja pozwala aplikacjom, takim jak serwery bazy danych, utrzymywać większe porcje bazy w obszarze adresów procesu. Dzięki temu zmniejsza się potrzeba mapowania na dysku widoków podzbiorów bazy danych, a tym samym zwiększa się ogólna wydajność (choć w niektórych przypadkach utrata 1 GB pamięci przez system może powodować bardziej wyraźne spadki wydajności w skali całego systemu). Na rysunku 1.4 pokazano dwa typowe układy obszaru adresów wirtualnych obsługiwane przez 32-bitowy system Windows (opcja `increaseuserva` umożliwia obrazom wykonywalnym oznaczonym za pomocą flagi dużego obszaru adresów zastosowanie pamięci o wielkości od 2 do 3 GB).



RYSUNEK 1.4. *Typowe układy obszaru adresów 32-bitowego systemu Windows*

Choć 3 GB jest lepsze niż 2 GB, w dalszym ciągu nie jest to obszar adresów wirtualnych wystarczający do mapowania bardzo dużych baz danych (liczących wiele gigabajtów). Aby umożliwić to w 32-bitowym systemie Windows, zapewniany jest przez niego mechanizm *AWE* (ang. *Address Windowing Extensions*), który pozwala 32-bitowej aplikacji przydzielić maksymalnie 64 GB pamięci fizycznej, a następnie mapować widoki lub okna na własny obszar adresów wirtualnych aplikacji o wielkości 2 GB. Choć użycie mechanizmu AWE powoduje obciążenie projektanta nakładem pracy związanym z zarządzaniem mapowaniem pamięci wirtualnej na fizyczną, daje możliwość uzyskania bezpośredniego dostępu do większej ilości pamięci fizycznej, niż może być mapowane w dowolnym momencie w 32-bitowym obszarze adresów procesu.

64-bitowy system Windows zapewnia procesom znacznie większy obszar adresów: 128 TB w wypadku systemów Windows 8.1, Server 2012 R2 oraz nowszych. Na rysunku 1.5 pokazano uproszczony widok układów 64-bitowych systemowych obszarów adresów (szczegółowy opis zamieszczono w rozdziale 5.). Zauważ, że podane wielkości nie reprezentują ograniczeń architektury obowiązujących dla tych platform. 64 bity obszaru adresów to 2 do potęgi 64. lub 16 EB (gdzie 1 EB równa się 1024 PB lub 1 048 576 TB), ale obecnie używane urządzenia 64-bitowe ograniczają to do mniejszych wartości. Niemapowany obszar oznaczony na rysunku 1.5 jest znacznie większy niż możliwy obszar mapowany (w systemie Windows 8 około miliona razy większy), co oznacza, że obrazy nie są (na razie) skalowane.



RYSUNEK 1.5. *Układy obszaru adresów 64-bitowego systemu Windows*

W rozdziale 5. opisano szczegóły implementacji menedżera pamięci, w tym sposób działania translacji adresów, a także to, jak system Windows zarządza pamięcią fizyczną.

Porównanie trybu jądra i trybu użytkownika

Aby uniemożliwić aplikacjom użytkownika dostęp do krytycznych danych systemu operacyjnego i/lub ich modyfikację, w systemie Windows stosowane są dwa tryby dostępu do procesora (jeśli nawet procesor, z którego korzysta system, obsługuje więcej niż dwa tryby): *tryb użytkownika* i *tryb jądra*. Kod aplikacji użytkownika działa w trybie użytkownika, natomiast kod systemu operacyjnego (np. usługi systemowe i sterowniki urządzeń) w trybie jądra. Tryb jądra odwołuje się do trybu wykonywania w procesorze, który zapewnia dostęp do całej pamięci systemowej i wszystkich instrukcji procesora. W wypadku części procesorów tryby takie są odróżniane za pomocą terminu *poziom przywilejów kodu* lub *poziom pierścienia*, natomiast w przypadku pozostałych są stosowane takie terminy, jak *tryb nadzorcy* i *tryb aplikacji*. Niezależnie od użytych terminów, dzięki zapewnieniu jądra systemu operacyjnego z wyższym poziomem przywilejów niż w przypadku aplikacji trybu użytkownika procesor dostarcza projektantom systemów niezbędną bazę do zapewnienia, że niepoprawnie działająca aplikacja nie może wpłynąć niekorzystnie na stabilność całego systemu.

Uwaga. Architektury procesorów x86 i x64 definiują cztery poziomy przywilejów (lub pierścienie), aby ochronić kod systemu i dane przed nadpisaniem nieumyślnie lub celowo przez kod o niższych przywilejach. Na potrzeby trybu jądra i trybu użytkownika system Windows używa odpowiednio poziomu przywilejów 0 (lub pierścienia 0) oraz poziomu przywilejów 3 (lub pierścienia 3). Powodem stosowania przez system Windows tylko dwóch poziomów jest to, że w niektórych architekturach sprzętowych, takich jak ARM obecnie i MIPS/Alpha w przeszłości, zaimplementowano zaledwie dwa poziomy przywilejów. Zdecydowanie się na całkowite minimum pozwoliło uzyskać efektywniejszą i bardziej przenośną architekturę, zwłaszcza z tego względu, że inne poziomy pierścieni architektury x86/x64 nie oferują takich samych gwarancji co pierścienie od 0 do 3.

Choć każdy proces systemu Windows ma własny obszar pamięci prywatnej, kod sterowników urządzeń i kod systemu operacyjnego w trybie jądra współużytkują pojedynczy obszar adresów wirtualnych. Każda strona w pamięci wirtualnej jest oznaczona w celu wskazania, w jakim trybie dostępu musi znajdować się procesor, aby mógł odczytać i/lub zapisać stronę. Strony w obszarze systemowym mogą być dostępne tylko w trybie jądra, natomiast wszystkie strony w obszarze adresów użytkownika są dostępne zarówno w trybie użytkownika, jak i w trybie jądra. Strony tylko do odczytu (zawierające na przykład dane statyczne) nie mogą być zapisywane w żadnym trybie. Ponadto w wypadku procesorów zabezpieczających przed atakami przepełnienia bufora pamięci system Windows oznacza strony zawierające dane jako niemożliwe do wykonania. W ten sposób system zapobiega nieumyślnemu lub celowemu wykonaniu kodu w obszarach danych (jeśli włączona jest funkcja DEP — *Data Execution Prevention*).

System Windows nie zapewnia żadnej ochrony dotyczącej prywatnej pamięci systemowej do odczytu i zapisu, która używana jest przez komponenty działające w trybie jądra. Inaczej mówiąc, w trybie jądra kod systemu operacyjnego i sterowników urządzeń ma pełny dostęp do pamięci obszaru systemowego, a ponadto przy uzyskiwaniu dostępu do obiektów może pominąć zabezpieczenia systemu Windows. Ponieważ większa część kodu tego systemu działa w trybie jądra, kluczowe znaczenie ma staranne projektowanie i testowanie komponentów uruchamianych w trybie jądra, aby zapewnić, że nie naruszą one bezpieczeństwa systemu lub nie spowodują jego niestabilności.

Taki brak ochrony uwidacznia też potrzebę zachowania czujności przy ładowaniu zewnętrznego sterownika urządzenia, a zwłaszcza wtedy, gdy nie jest on podpisany, ponieważ w trybie jądra sterownik dysponuje pełnym dostępem do wszystkich danych systemu operacyjnego. Ryzyko to było jednym z powodów wprowadzenia w systemie Windows 2000 mechanizmu podpisywania sterowników, który ostrzega (i blokuje po przeprowadzeniu odpowiedniej konfiguracji) użytkownika, jeśli podejmowana jest próba dodania sterownika Plug and Play bez podpisu (więcej informacji o podpisywaniu sterowników zamieszczono w rozdziale 6., „System operacji wejścia-wyjścia"). Mechanizm ten nie wpływa jednak na inne typy sterowników. Poza tym, mechanizm Driver Verifier ułatwia twórcom sterowników urządzeń znajdowanie błędów, takich jak nadpisania bufora lub wycieki pamięci, które mogą spowodować problemy z zabezpieczeniami lub niezawodnością (mechanizm ten również omówiono w rozdziale 6.).

W przypadku 64-bitowych wersji systemu Windows 8.1 oraz jego wersji dla urządzeń ARM zasada KMCS (ang. *Kernel-Mode Code-Signing*) podpisywania kodu w trybie jądra wymusza konieczność podpisywania wszystkich sterowników urządzeń (a nie tylko sterowników Plug and Play) przy użyciu klucza kryptograficznego przydzielonego przez jeden z głównych urzędów certyfikacji kodu. Użytkownik nie może wprost wymusić instalacji niepodpisanego sterownika, nawet jako administrator. Jednak w ramach jednorazowego wyjątku ograniczenie to może zostać ręcznie wyłączone. Umożliwia to automatyczne podpisanie i testowanie sterowników, a ponadto powoduje umieszczenie na tapecie pulpitu znaku wodnego informującego o trybie testowym i wyłączenie określonych funkcji zarządzania prawami cyfrowymi (DRM — *Digital Rights Management*).

W systemie Windows 10 firma Microsoft wprowadziła jeszcze bardziej znaczącą zmianę, która została wymuszona rok po udostępnieniu jako część lipcowej aktualizacji Anniversary Update (wersja 1607). Od tego czasu wszystkie nowe sterowniki dla systemu Windows 10 muszą być podpisane tylko przez dwa akceptowane urzędy certyfikacji za pomocą certyfikatu SHA-2 Extended Validation (EV), a nie przy użyciu zwykłego certyfikatu SHA-1 opartego na pliku oferowanego przez 20 urzędów. Sterownik urządzenia podpisany za pomocą certyfikatu EV musi zostać przesłany firmie Microsoft za pośrednictwem portalu System Device (SysDev) w celu poświadczenia podpisu i otrzymania podpisu Microsoftu. W tej sytuacji jądro będzie podpisywać wyłącznie sterowniki dla systemu Windows 10 podpisane przez firmę Microsoft bez żadnych wyjątków, pomijając wcześniej wspomniany tryb testowy. Sterowniki podpisane przed datą opublikowania systemu Windows 10 (lipiec 2015 r.) mogą na razie być nadal ładowane z użyciem zwykłego podpisu.

Jak dotąd w wypadku systemu operacyjnego Windows Server 2016 podjęto najbardziej stanowczą decyzję. Bazowanie na wcześniej podanych wymaganiach w postaci certyfikatów EV sprawia, że samo podpisywanie poświadczenia staje się niewystarczające. Aby sterownik dla systemu Windows 10 został załadowany w systemie serwera, musi przejść rygorystyczny proces certyfikacji WHQL (ang. *Windows Hardware Quality Labs*) jako część pakietu HCK (ang. *Hardware Compatibility Kit*), a ponadto musi zostać przesłany do formalnej oceny. Możliwość załadowania w takich systemach będą mieć tylko sterowniki podpisane za pomocą certyfikatu WHQL, które zapewniają administratorom systemu określone gwarancje dotyczące zgodności, bezpieczeństwa, wydajności i stabilności. Podsumowując, ograniczenie liczby zewnętrznych sterowników, które mogą zostać załadowane w pamięci trybu jądra, powinno spowodować znaczne zwiększenie poziomu stabilności i bezpieczeństwa.

W wypadku konkretnych dostawców i platform, a nawet konfiguracji systemu Windows dla przedsiębiorstw może istnieć dowolna liczba dostosowanych zasad podpisywania, takich jak oferowane przez technologię Device Guard, która w skrócie zostanie opisana w następnym punkcie, „Hipernadzorca", a później w rozdziale 7. W związku z tym przedsiębiorstwo może

wymagać podpisów WHQL nawet w systemach klienckich Windows 10 albo może zażądać pominięcia tego wymogu w systemie Windows Server 2016.

Jak się okaże w rozdziale 2., „Architektura systemu", aplikacje użytkownika dokonują przełączenia z trybu użytkownika do trybu jądra w momencie tworzenia wywołania usługi systemowej. Na przykład funkcja `ReadFile` systemu Windows wymaga ostatecznie wywołania jego wewnętrznego programu, który w rzeczywistości obsługuje odczyt danych z pliku. Ponieważ program ten uzyskuje dostęp do wewnętrznych struktur danych systemu, musi działać w trybie jądra. Użycie specjalnej instrukcji procesora wyzwala przejście z trybu użytkownika do trybu jądra, a ponadto powoduje aktywowanie przez procesor usługi systemowej rozdzielającej kod w jądrze. To z kolei powoduje wywołanie odpowiedniej funkcji wewnętrznej w pliku *Ntoskrnl.exe* lub *Win32k.sys*. Przed przekazaniem kontroli wątkowi użytkownika tryb procesora z powrotem przełączany jest na tryb użytkownika. W ten sposób system operacyjny chroni siebie i swoje dane przed wglądem procesów użytkownika i modyfikowaniem.

Uwaga. Przejście z trybu użytkownika do trybu jądra (i z powrotem) nie wpływa na planowanie wątków *jako takie*. Przejście trybów *nie* jest przełączaniem kontekstu. W rozdziale 2. zamieszczono dodatkowe szczegóły dotyczące rozdzielania usług systemowych.

A zatem normalną sytuacją jest to, że wątek użytkownika spędza część swojego czasu wykonywania w trybie użytkownika, a część w trybie jądra. Okazuje się, że ponieważ spora część systemu graficznego i systemu obsługującego okna również działa w trybie jądra, aplikacje intensywnie korzystające z grafiki więcej czasu spędzają w trybie jądra niż w trybie użytkownika. Prostym sposobem sprawdzenia tego jest uruchomienie tego rodzaju aplikacji, takiej jak Microsoft Paint, i obserwowanie za pomocą liczników wydajności (wyszczególnionych w tabeli 1.3) podziału czasu między trybem użytkownika i trybem jądra. Bardziej zaawansowane aplikacje mogą używać nowszych technologii, takich jak Direct2D i DirectComposition, które większość obliczeń wykonują w trybie użytkownika, a do jądra kierują tylko nieprzetworzone dane związane z powierzchniami. Dzięki temu skraca się czas trwania operacji przejścia między trybem użytkownika i trybem jądra.

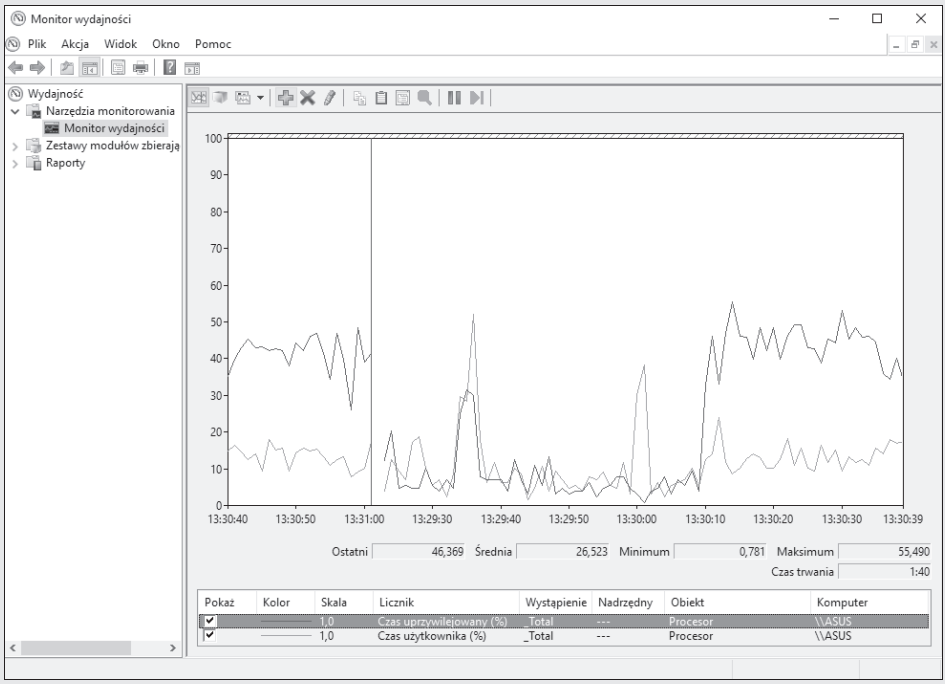
TABELA 1.3. *Liczniki wydajności powiązane z trybami*

Obiekt: licznik	Funkcja
Procesor: Czas uprzywilejowany (%)	Wartość procentowa czasu, przez jaki pojedynczy procesor (lub wszystkie procesory) musi działać w trybie jądra podczas określonego interwału.
Procesor: Czas użytkownika (%)	Wartość procentowa czasu, przez jaki pojedynczy procesor (lub wszystkie procesory) musi działać w trybie użytkownika podczas określonego interwału.
Proces: Czas uprzywilejowany (%)	Wartość procentowa czasu, przez jaki wątki w procesie muszą działać w trybie jądra podczas określonego interwału.
Proces: Czas użytkownika (%)	Wartość procentowa czasu, przez jaki wątki w procesie muszą działać w trybie użytkownika podczas określonego interwału.
Wątek: Czas uprzywilejowany (%)	Wartość procentowa czasu, przez jaki wątek musi działać w trybie jądra podczas określonego interwału.
Wątek: Czas użytkownika (%)	Wartość procentowa czasu, przez jaki wątek musi działać w trybie użytkownika podczas określonego interwału.

EKSPERYMENT: Porównanie trybu użytkownika i trybu jądra

Za pomocą narzędzia Monitor wydajności możesz sprawdzić, ile czasu system spędza odpowiednio na działaniu w trybie jądra i trybie użytkownika. Wykonaj następujące kroki:

1. Wyświetl menu *Start* i wpisz **`Monitor wydajności`** (polecenie powinno zostać zasugerowane przed zakończeniem wpisywania go), aby uruchomić Monitor wydajności.
2. Wybierz węzeł *Monitor wydajności* poniżej węzła *Wydajność/Narzędzia monitorowania* w drzewie po lewej stronie.
3. W celu usunięcia domyślnego licznika pokazującego całkowity czas procesora kliknij przycisk *Usuń* na pasku narzędzi lub naciśnij klawisz *Delete* klawiatury.
4. Kliknij przycisk *Dodaj (+)* na pasku narzędzi.
5. Rozwiń sekcję licznika *Procesor*, kliknij licznik *Czas uprzywilejowany (%)*, a następnie, trzymając wciśnięty klawisz *Ctrl*, kliknij licznik *Czas użytkownika (%)*.
6. Kliknij kolejno przyciski *Dodaj* i *OK*.
7. Otwórz okno wiersza poleceń i wpisz polecenie **`dir \\%computername%\c$ /s`**, aby uruchomić skanowanie katalogów dysku *C*.



8. Po zakończeniu operacji zamknij narzędzie.

Te same informacje możesz uzyskać też za pomocą Menedżera zadań. Wystarczy kliknąć kartę *Wydajność*, prawym przyciskiem myszy kliknąć wykres procesora i wybrać opcję *Pokaż czas jądra*. Na pasku wykorzystania procesora pojawi się czas spędzony przez procesor w trybie jądra wyróżniony ciemniejszym odcieniem koloru jasnoniebieskiego.

Aby sprawdzić, ile czasu działa sam Monitor wydajności w trybie jądra i trybie użytkownika, ponownie uruchom to narzędzie, lecz dla każdego procesu w systemie dodaj osobne liczniki procesu *Czas użytkownika (%)* i *Czas uprzywilejowany (%)*:

1. Jeśli narzędzie Monitor wydajności jeszcze nie działa, uruchom je ponownie (jeżeli już działa, rozpocznij od pustego widoku, klikając prawym przyciskiem myszy w obszarze wykresu i wybierając pozycję *Usuń wszystkie liczniki*).
2. Kliknij przycisk *Dodaj* na pasku narzędzi.
3. W dostępnym obszarze liczników rozwiń sekcję *Proces*.
4. Wybierz liczniki *Czas uprzywilejowany (%)* i *Czas użytkownika (%)*.
5. W polu *Wystąpienia wybranego obiektu* wybierz kilka procesów (np. *mmc*, *csrss* i *Idle*).
6. Kliknij kolejno przyciski *Dodaj* i *OK*.
7. Przesuń szybko kursor myszy do tyłu i do przodu.
8. Użyj kombinacji klawiszy *Ctrl+H*, aby włączyć tryb wyróżniania. Spowoduje to wyróżnienie na czarno aktualnie wybranego licznika.
9. Przewiń liczniki do dołu widoku, aby zidentyfikować procesy, których wątki zostały uruchomione w momencie przesunięcia kursora myszy. Zwróć uwagę, czy zostały one uruchomione w trybie użytkownika, czy w trybie jądra.

Po przesunięciu kursora myszy powinno być widoczne w oknie narzędzia Monitor wydajności, że w kolumnie *Wystąpienie* procesu *mmc* zwiększa się czas dla trybu jądra *i* trybu użytkownika. Wynika to stąd, że proces wykonuje kod aplikacji w trybie użytkownika i wywołuje funkcje systemu Windows, które działają w trybie jądra. Zauważysz również aktywność wątku w trybie jądra w procesie o nazwie *csrss* w momencie przesunięcia kursora myszy. Powodem tej aktywności jest to, że do tego procesu dołączony jest związany z podsystemem systemu Windows wątek nieprzetworzonych danych wejściowych trybu jądra, który obsługuje dane wejściowe myszy i klawiatury (w rozdziale 2. zamieszczono więcej informacji o wątkach systemowych i podsystemach). Proces *Idle*, który niemal 100% swojego czasu spędza w trybie jądra, w rzeczywistości nie jest procesem. Jest to fałszywy proces używany do uwzględniania bezczynnych cykli procesora. Jak możesz zaobserwować na podstawie trybu, w jakim działają wątki procesu *Idle*, gdy system Windows nie ma żadnych operacji do wykonania, działa w trybie jądra.

Hipernadzorca

Ostatnie zmiany w modelach aplikacji i oprogramowania, takie jak wprowadzenie usługi w chmurze oraz wszechobecność urządzeń IoT, spowodowały konieczność określenia przez dostawców systemów operacyjnych i sprzętu bardziej efektywnych sposobów wirtualizacji innych „gości” systemu z wykorzystaniem urządzeń komputera będącego hostem. Jest to niezbędne niezależnie od tego, czy ma na celu umożliwienie udostępnienia wielu dzierżawców w farmie serwerów i uruchomienia na pojedynczym serwerze 100 izolowanych witryn internetowych, czy zapewnienie projektantom możliwości testowania dziesiątek różnych wariantów systemu operacyjnego bez konieczności kupowania dedykowanego sprzętu. Potrzeba zastosowania szybkiej, efektywnej i bezpiecznej wirtualizacji przyczyniła się do pojawienia się nowych związanych z oprogramowaniem modeli dotyczących przeprowadzania obliczeń i logiki.

Okazuje się, że obecnie określone oprogramowanie, takie jak Docker, który obsługiwany jest przez systemy Windows 10 i Windows Server 2016, działa w kontenerach zapewniających w pełni izolowane maszyny wirtualne zaprojektowane wyłącznie pod kątem uruchamiania stosu lub środowiska jednej aplikacji. Oznacza to, że granice dotyczące „gościa" i hosta zostały przesunięte jeszcze dalej.

W celu zapewnienia takich usług wirtualizacji w niemal wszystkich nowoczesnych rozwiązaniach zastosowano *hipernadzorcę* (ang. *hypervisor*), czyli specjalizowany komponent o bardzo dużych przywilejach, który umożliwia wirtualizację i izolowanie wszystkich zasobów komputera, począwszy od pamięci wirtualnej, a skończywszy na pamięci fizycznej, przerwaniach sprzętowych, a nawet urządzeniach PCI i USB. Hyper-V to przykład takiego hipernadzorcy, który obsługuje funkcjonalność klienta Hyper-V dostępną w systemie Windows 8.1 oraz jego następcach. W konkurencyjnych produktach, takich jak Xen, KVM, VMware i VirtualBox, implementowany jest ich własny hipernadzorca. Każdy z nich ma swoje słabe i mocne strony.

Ze względu na swoje szczególnie duże przywileje i to, że hipernadzorca dysponuje jeszcze większym poziomem dostępu niż samo jądro, zapewnia on wyraźną korzyść, która wykracza poza samo uruchamianie wielu instancji „gości" innych systemów operacyjnych. Hipernadzorca może chronić i monitorować pojedynczą instancję hosta w celu zapewnienia gwarancji większych niż oferowane przez jądro. W systemie Windows 10 firma Microsoft stosuje obecnie hipernadzorcę Hyper-V, aby udostępnić nowy zestaw mechanizmów *zabezpieczeń VBS* (ang. *Virtualization-Based Security*). Oto one:

- **Device Guard**. Mechanizm zapewnia usługę HVCI (ang. *Hypervisor Code Integrity*) w celu zaoferowania lepszych gwarancji podpisywaniu kodu niż w porównaniu z samą zasadą KMCS. Mechanizm umożliwia dostosowywanie zasady podpisów systemu Windows, zarówno w wypadku trybu użytkownika, jak i trybu jądra.
- **Hyper Guard**. Mechanizm chroni kluczowe struktury danych i kod powiązane z jądrem i hipernadzorcą.
- **Credential Guard**. Mechanizm zapobiega nieautoryzowanemu dostępowi do danych uwierzytelniających i haseł kont domenowych, a ponadto oferuje bezpieczną technologię biometryczną.
- **Application Guard**. Mechanizm zapewnia jeszcze lepszą „piaskownicę" na potrzeby przeglądarki Microsoft Edge.
- **Host Guardian** i **Shielded Fabric**. Mechanizmy wykorzystują funkcję v-TPM (ang. *virtual TPM*) do ochrony maszyny wirtualnej przed infrastrukturą, w wypadku której ją uruchomiono.

Ponadto hipernadzorca Hyper-V pozwala za zastosowanie określonych, kluczowych ograniczeń jądra względem programów wyszukujących luki w zabezpieczeniach (ang. *exploits*) oraz innych rodzajów ataków. Kluczową zaletą wszystkich tych technologii jest to, że w przeciwieństwie do wcześniejszych ulepszeń zabezpieczeń opartych na jądrze, nie są one podatne na złośliwe lub źle napisane sterowniki, niezależnie od tego, czy są one podpisane, czy nie. Dzięki temu technologie te są wysoce odporne na obecnie istniejące zaawansowane rozwiązania tworzone przez osoby podejmujące ataki. Jest to możliwe z powodu zaimplementowania przez hipernadzorcę wirtualnych poziomów zaufania VTL (ang. *Virtual Trust Level*). Ponieważ standardowy system operacyjny i jego komponenty działają w mniej uprzywilejowanym trybie (VTL 0), a wymienione technologie VBS pracują w trybie VTL 1 (większe przywileje), wpływu na nie może nie mieć nawet kod w trybie jądra. W związku z tym kod pozostaje w obrębie obszaru z przywilejami trybu VTL 0. Tym sposobem poziomy VTL możesz traktować jako ortogonalne względem poziomów przywilejów procesora:

tryb jądra i tryb użytkownika istnieją *wewnątrz* każdego poziomu VTL, a hipernadzorca zarządza przywilejami *między* poziomami VTL. W rozdziale 2. zamieszczono dodatkowe szczegóły dotyczące architektury wspomaganej przez hipernadzorcę, a w rozdziale 7. omówiono dokładnie zaprezentowane wcześniej mechanizmy zabezpieczeń VBS.

Oprogramowanie sprzętowe

Komponenty systemu Windows w coraz większym stopniu są zależne od bezpieczeństwa systemu operacyjnego i jego jądra. Z kolei jądro bazuje obecnie na ochronie hipernadzorcy. Pojawia się pytanie odnośnie tego, co może zapewnić, że komponenty te są bezpiecznie ładowane, a ponadto mogą uwierzytelniać swoją zawartość. Jest to zwykle zadanie rozruchowego programu ładującego, który jednak również wymaga takiego samego poziomu sprawdzania uwierzytelniania, co powoduje utworzenie coraz bardziej złożonej hierarchii zaufania.

Co zatem zapewnia główny łańcuch zaufania, który może zagwarantować nieograniczony niczym proces rozruchu? W nowoczesnym systemie Windows 8 i jego następcach kwestia ta podlega systemowemu oprogramowaniu sprzętowemu, które w certyfikowanych systemach musi bazować na interfejsie UEFI (ang. *Unified Extensible Firmware Interface*). Bycie częścią standardu UEFI, który narzucany jest w systemie Windows (UEFI 2.3.1b; więcej informacji dostępnych jest pod adresem *http://www.uefi.org/*), oznacza konieczność obecności bezpiecznej implementacji rozruchu z silnymi gwarancjami i wymaganiami dotyczącymi jakości podpisu oprogramowania powiązanego z procesem rozruchu. Dzięki takiemu procesowi weryfikacji uzyskuje się gwarancję bezpiecznego ładowania komponentów systemu Windows od samego początku procesu rozruchu. Dodatkowo technologie takie jak TPM (ang. *Trusted Platform Module*) mogą dokonywać pomiaru procesu w celu zapewnienia poświadczenia (zarówno lokalnie, jak i zdalnie). Dzięki partnerstwu z przedstawicielami branży firma Microsoft zarządza „białą” i „czarną” listą zgodnych ze standardem UEFI komponentów bezpiecznego rozruchu w przypadku błędów lub luk w zabezpieczeniach oprogramowania rozruchowego. Aktualizacje systemu Windows obejmują też obecnie aktualizacje oprogramowania sprzętowego (warto w tym miejscu podkreślić jego znaczenie w nowoczesnej architekturze systemu Windows z punktu widzenia gwarancji, jakie ma zapewnić).

Usługi terminalowe i wiele sesji

Usługi terminalowe odnoszą się do obsługi w pojedynczym systemie Windows wielu interaktywnych sesji użytkownika. Dzięki usługom terminalowym systemu Windows użytkownik zdalny może ustanowić sesję na innym komputerze, zalogować się i uruchomić aplikacje na serwerze. Serwer przekazuje do klienta graficzny interfejs użytkownika (a także inne konfigurowalne zasoby, takie jak dźwięk i schowek). Klient transmituje dane wprowadzone przez użytkownika z powrotem do serwera (podobnie do środowiska X Window System, system Windows umożliwia uruchamianie poszczególnych aplikacji w systemie serwera, gdzie zamiast zdalnego przesyłania zawartości całego pulpitu dane są wyświetlane zdalnie po stronie klienta).

Pierwsza sesja jest uważana za sesję usług lub sesję zerową. Zawiera ona usługę systemową udostępniającą procesy. Pierwsza sesja logowania w konsoli fizycznej komputera to sesja 1. Dodatkowe sesje mogą być tworzone z wykorzystaniem programu Podłączanie pulpitu zdalnego (plik *Mstsc.exe*) lub funkcji szybkiego przełączania użytkowników.

Wersje klienckie systemu Windows zezwalają na nawiązanie połączenia z komputerem przez pojedynczego użytkownika zdalnego. Jeśli jednak ktoś jest zalogowany w konsoli, stacja robocza jest zablokowana. Oznacza to, że ktoś może korzystać z systemu lokalnie lub zdalnie, lecz nie jednocześnie. Wersje systemu Windows obejmujące wersję Windows Media Center umożliwiają jedną sesję interaktywną i maksymalnie cztery sesje urządzenia Windows Media Center Extender.

Systemy serwerowe Windows obsługują dwa jednoczesne połączenia zdalne. Ma to na celu ułatwienie zarządzania zdalnego (na przykład zastosowanie narzędzi zarządzania wymagających zalogowania na zarządzanym komputerze). Systemy te wspierają też więcej niż dwie sesje zdalne, jeśli zostaną odpowiednio skonfigurowane jako serwer terminali, a ponadto dysponują właściwą licencją.

Wszystkie wersje klienckie systemu Windows obsługują wiele sesji tworzonych lokalnie za pomocą funkcji *szybkiego przełączania użytkowników*. W danym momencie może być używana tylko jedna sesja. Gdy użytkownik zdecyduje się na rozłączenie swojej sesji, a nie wylogowanie (na przykład przez kliknięcie przycisku *Start* i ikony bieżącego użytkownika, a następnie wybranie z wyświetlonego podmenu pozycji *Przełącz konto* albo przez naciśnięcie klawisza *L* przy wciśniętym klawiszu logo systemu Windows i kliknięcie innego użytkownika w dolnym lewym narożniku ekranu), bieżąca sesja, czyli procesy działające w danej sesji oraz wszystkie struktury danych sesji opisujące ją, pozostaje aktywna w systemie, który przywraca główny ekran logowania (jeśli nie jest jeszcze widoczny). W przypadku zalogowania nowego użytkownika zostanie utworzona nowa sesja.

W wypadku aplikacji, które mają rozpoznawać fakt działania w sesji serwera terminali, dostępny jest zestaw interfejsów API systemu Windows służących do programowego wykrywania tego, a także do kontrolowania różnych aspektów usług terminalowych (związane z tym szczegóły znajdziesz w dokumentacji zestawu SDK systemu Windows i interfejsu API usług pulpitu zdalnego).

W rozdziale 2. w skrócie opisano sposób tworzenia sesji. Ponadto zamieszczono w nim kilka eksperymentów prezentujących, jak za pomocą różnych narzędzi (w tym debuggera jądra) wyświetlać informacje o sesjach. W rozdziale 5. przybliżono, jak menedżer pamięci przygotowuje dane całej sesji oraz nimi zarządza.

Obiekty i dojścia

W systemie operacyjnym Windows *obiekt jądra* to pojedyncza wykonawcza instancja statycznie zdefiniowanego typu obiektu. *Typ obiektu* składa się z typu danych definiowanego w systemie, funkcji przetwarzających instancje typu danych oraz zestawu atrybutów obiektu. Jeśli tworzysz aplikacje dla systemu Windows, możesz napotkać obiekty procesu, wątku, pliku i zdarzenia. A to tylko kilka przykładów. Obiekty te bazują na obiektach niższego poziomu tworzonych i zarządzanych przez system Windows. W systemie tym *proces* to instancja typu obiektu procesu, *plik* to instancja typu obiektu pliku itd.

Atrybut obiektu jest polem danych w obiekcie, które częściowo definiuje jego stan. Na przykład obiekt typu *proces* będzie mieć atrybuty, które obejmują identyfikator procesu, podstawowy priorytet planowania oraz wskaźnik do obiektu tokena dostępu. *Metody obiektu*, czyli środki umożliwiające manipulowanie obiektami, zazwyczaj wczytują lub zmieniają atrybuty obiektu. Na przykład metoda *otwierająca* procesu będzie akceptować identyfikator procesu jako dane wejściowe i zwracać wskaźnik do obiektu jako dane wyjściowe.

Uwaga. Istnieje parametr o nazwie `ObjectAttributes`, który jest dostarczany przez element wywołujący podczas tworzenia obiektu za pomocą interfejsów API menedżera obiektów jądra. Parametru tego nie należy jednak mylić z bardziej ogólnym znaczeniem terminu używanym w książce.

Najbardziej fundamentalną różnicą między obiektem i zwykłą strukturą danych jest to, że struktura wewnętrzna obiektu jest nieprzejrzysta. W celu uzyskania danych z obiektu lub umieszczenia w nim danych niezbędne jest wywołanie usługi obiektu. Nie możesz bezpośrednio wczytać lub zmodyfikować danych wewnątrz obiektu. Różnica ta oddziela bazową implementację obiektu od kodu, który tylko z niego korzysta. Jest to technika umożliwiająca łatwe modyfikowanie z upływem czasu implementacji obiektu.

Dzięki wsparciu komponentu jądra nazywanego *menedżerem obiektów* obiekty zapewniają wygodne metody realizowania czterech następujących ważnych zadań systemu operacyjnego:

- Zapewnianie czytelnych dla użytkownika nazw zasobów systemowych.
- Współużytkowanie zasobów i danych przez procesy.
- Ochrona zasobów przed nieautoryzowanym dostępem.
- Śledzenie odwołań, które umożliwia systemowi określanie momentu, kiedy obiekt nie jest już dłużej używany, dlatego może zostać dla niego wykonana automatycznie operacja usuwania alokacji.

Nie wszystkie struktury danych w systemie operacyjnym Windows są obiektami. W obiektach umieszczane są tylko te dane, które wymagają współużytkowania, ochrony, nadania nazwy lub udostępnienia programom w trybie użytkownika (za pośrednictwem usług systemowych). Obiektami nie są struktury stosowane przez tylko jeden komponent systemu w celu implementacji funkcji wewnętrznych.

Zabezpieczenia

System Windows od początku zaprojektowano tak, aby był bezpieczny i spełniał wymagania różnych formalnych, zarówno rządowych, jak i branżowych wskaźników bezpieczeństwa, takich jak specyfikacja CCITSE (ang. *Common Criteria for Information Technology Security Evaluation*). Uzyskanie wskaźnika bezpieczeństwa akceptowanego na szczeblu rządowym pozwala systemowi operacyjnemu konkurować w tym obszarze. Oczywiście wiele związanych z tym możliwości to korzystne funkcje dowolnego systemu obsługującego wielu użytkowników.

Podstawowe możliwości systemu Windows związane z bezpieczeństwem obejmują następujące:

- Nieokreślona (na żądanie) i obowiązkowa ochrona wszystkich obiektów systemowych, które mogą być współużytkowane, takich jak pliki, katalogi, procesy, wątki itd.
- Audyt zabezpieczeń w celu umożliwienia zestawienia podmiotów lub użytkowników oraz inicjowanych przez nich działań.
- Uwierzytelnianie użytkownika podczas logowania.
- Uniemożliwienie jednemu użytkownikowi uzyskania dostępu do niezainicjowanych zasobów, takich jak wolna pamięć lub przestrzeń dyskowa, które po przydzieleniu zostały zwrócone przez drugiego użytkownika.

W systemie Windows istnieją trzy następujące formy kontroli dostępu do obiektów:

- **Uznaniowa kontrola dostępu**. Jest to mechanizm ochrony, który ma na myśli większość osób zajmujących się zabezpieczeniami systemu operacyjnego. W wypadku tej metody dostępu właściciele obiektów (np. pliki lub drukarki) udzielają dostępu innym lub go odbierają. W momencie zalogowania użytkownicy uzyskują zestaw danych uwierzytelniających zabezpieczeń lub kontekst zabezpieczeń. Przy próbie uzyskania przez użytkowników dostępu do obiektów ich kontekst zabezpieczeń porównywany jest z listą kontroli dostępu obiektu, dla którego ma miejsce próba uzyskania dostępu. Ma to na celu stwierdzenie, czy użytkownicy mają uprawnienie do wykonania żądanej operacji. W przypadku systemów Windows Server 2012 i Windows 8 taką formę kontroli uznaniowej dodatkowo ulepszono przez zaimplementowanie kontroli dostępu opartej na atrybutach (nazywanej również dynamiczną kontrolą dostępu). Lista kontroli dostępu zasobu niekoniecznie jednak identyfikuje poszczególnych użytkowników i grupy. Zamiast tego identyfikuje wymagane atrybuty lub uprawnienia nadające dostęp do zasobu (np. „Poziom zezwolenia: ściśle tajne" lub „Staż pracy: 10 lat"). Dzięki możliwości automatycznego wprowadzania takich atrybutów przez analizowanie baz danych SQL i schematów za pośrednictwem usługi Active Directory ten znacznie bardziej elegancki i elastyczny model zabezpieczeń ułatwia organizacjom unikanie kłopotliwych hierarchii grup oraz ręcznego zarządzania grupami.
- **Uprzywilejowana kontrola dostępu**. Ta forma kontroli jest niezbędna w sytuacjach, gdy uznaniowa kontrola dostępu jest niewystarczająca. Jest to metoda zapewniania, że ktoś może uzyskać dostęp do chronionych obiektów, jeśli ich właściciel jest niedostępny. Jeśli na przykład pracownik opuści firmę, administrator musi dysponować możliwością uzyskania dostępu do plików, które mogły być dostępne tylko dla tego pracownika. W tym przypadku w systemie Windows administrator może przejąć prawa właściciela pliku, dzięki czemu w razie potrzeby może zarządzać jego prawami.
- **Obowiązkowa kontrola dostępu**. Ta forma kontroli jest wymagana, gdy niezbędny jest dodatkowy poziom kontroli zabezpieczeń w celu ochrony obiektów, do których uzyskiwany jest dostęp z poziomu tego samego konta użytkownika. Obowiązkowa kontrola dostępu jest stosowana w każdym przypadku, począwszy od zapewniania części technologii „piaskownicy" aplikacjom Windows Apps (więcej na ten temat w dalszej części rozdziału), a skończywszy na izolowaniu trybu chronionego przeglądarki Internet Explorer (oraz innych przeglądarek) od konfiguracji użytkownika oraz ochronie obiektów utworzonych przez administratora z większymi uprawnieniami przed dostępem z poziomu konta administratora o mniejszych uprawnieniach (w rozdziale 7. zamieszczono więcej informacji o komponencie Kontrola konta użytkownika).

Począwszy od systemu Windows 8, do udostępniania aplikacji Windows Apps używana jest „piaskownica" o nazwie *AppContainer*, która zapewnia izolację w odniesieniu do innych komponentów AppContainer oraz procesów niepowiązanych z aplikacjami Windows Apps. Kod w komponentach AppContainer może komunikować się z brokerami (nieizolowane procesy działające z danymi uwierzytelniającymi użytkownika), a czasami z innymi komponentami AppContainer lub procesami za pośrednictwem dobrze zdefiniowanych kontraktów udostępnianych przez architekturę Windows Runtime. Kanonicznym przykładem jest przeglądarka Microsoft Edge, która działa w obrębie komponentu AppContainer, a tym samym zapewnia lepszą ochronę przed złośliwym kodem uruchomionym w granicach tego komponentu. Ponadto projektanci zewnętrznego oprogramowania mogą wykorzystać komponenty AppContainer do izolowania w podobny sposób ich własnych aplikacji, które nie są aplikacjami Windows Apps. Model komponentów

AppContainer wymusza znaczną zmianę w tradycyjnych modelach programowania, powodując przejście z klasycznej implementacji wielowątkowej aplikacji z pojedynczym procesem do aplikacji z wieloma procesami.

Zabezpieczenia obejmują interfejs API systemu Windows. Jego podsystem implementuje zabezpieczenia oparte na obiektach w taki sam sposób, w jaki realizuje to sam system operacyjny, czyli przez ochronę współużytkowanych obiektów systemu Windows przed nieautoryzowanym dostępem, co sprowadza się do zastosowania dla nich systemowych deskryptorów zabezpieczeń. Gdy aplikacja po raz pierwszy próbuje uzyskać dostęp do współużytkowanego obiektu, podsystem systemu Windows sprawdza, czy prawo aplikacji na to zezwala. Jeśli kontrola zabezpieczeń się powiedzie, podsystem umożliwia aplikacji kontynuowanie działania.

Obszerne omówienie zabezpieczeń systemu Windows znajdziesz w rozdziale 7.

Rejestr

Jeśli korzystałeś z systemów operacyjnych Windows, prawdopodobnie słyszałeś o rejestrze lub sprawdzałeś jego zawartość. Nie możesz zbyt obszernie dyskutować o wewnętrznych mechanizmach systemu Windows bez odwoływania się do rejestru, ponieważ jest to systemowa baza danych zawierająca informacje wymagane do rozruchu i konfiguracji systemu, systemowe ustawienia oprogramowania kontrolujące działanie systemu Windows, bazę danych zabezpieczeń oraz ustawienia konfiguracyjne poszczególnych użytkowników (określające na przykład, jaki wygaszacz ekranu ma zostać użyty). Ponadto rejestr zapewnia wgląd w ulotne dane znajdujące się w pamięci, takie jak bieżący stan urządzenia systemu (jakie sterowniki urządzeń są ładowane, używane przez nie zasoby itd.), a także liczniki wydajności systemu Windows. Liczniki te, które w rzeczywistości nie znajdują się *w* rejestrze, mogą być dostępne za pośrednictwem jego funkcji (choć istnieje nowszy i lepszy interfejs API zapewniający dostęp do liczników wydajności).

Choć wielu użytkowników i administratorów systemu Windows nigdy nie będzie mieć potrzeby bezpośredniego sprawdzania zawartości rejestru (ponieważ większość ustawień konfiguracyjnych może być wyświetlana lub modyfikowana za pomocą standardowych narzędzi administracyjnych), w dalszym ciągu jest to przydatne źródło informacji o wewnętrznych mechanizmach systemu Windows. Wynika to z tego, że rejestr zawiera wiele ustawień mających wpływ na wydajność i zachowanie systemu. W różnych miejscach książki napotkasz odwołania do poszczególnych kluczy rejestru, gdy będą odnosić się one do opisywanego komponentu. Większość kluczy rejestru przywoływanych w książce znajduje się w gałęzi konfiguracji całego systemu *HKEY_LOCAL_MACHINE*, dla której będzie używany skrót *HKLM*.

Ostrzeżenie. Jeśli zdecydujesz się na bezpośrednie wprowadzanie zmian w ustawieniach rejestru, musisz zachować szczególną ostrożność. Wszelkie zmiany mogą niekorzystnie wpłynąć na wydajność systemu lub, co gorsza, uniemożliwić pomyślny rozruch systemu.

Unicode

System Windows różni się od większości innych systemów operacyjnych tym, że większość wewnętrznych łańcuchów tekstowych przechowywana i przetwarzana jest jako 16-bitowe znaki Unicode (dokładnie rzecz biorąc, znaki UTF-16LE; gdy w książce jest mowa o standardzie Unicode, dotyczy to wersji UTF-16LE, chyba że wskazano inaczej). Unicode to międzynarodowy

standard zestawu znaków, który definiuje unikatowe wartości dla większości znanych na świecie zestawów znaków. Dla każdego znaku standard zapewnia 8-bitowe, 16-bitowe, a nawet 32-bitowe kodowania.

Ponieważ wiele aplikacji ma do czynienia z 8-bitowymi (jednobajtowymi) łańcuchami znaków ANSI, wiele funkcji systemu Windows, które akceptują parametry łańcuchowe, używa dwóch punktów wejścia: wersji Unicode („szeroka", 16-bitowa) oraz wersji ANSI („wąska", 8-bitowa). Jeśli wywołasz „wąską" wersję funkcji systemu Windows, będzie to mieć niewielki wpływ na wydajność, ponieważ przed zwróceniem aplikacji wejściowe parametry łańcuchowe są przekształcane z formatu Unicode do formatu ANSI. Jeśli zatem używasz starszej usługi lub porcji kodu wymagających uruchomienia w systemie Windows, ale kod ten napisano z wykorzystaniem łańcuchów tekstowych ze znakami ANSI, na własny użytek system dokona przekształcenia znaków ANSI do formatu Unicode. System Windows nigdy jednak nie konwertuje *danych* wewnątrz plików. Od aplikacji zależne jest to, czy dane zostaną zapisane w formacie Unicode, czy w formacie ANSI.

Niezależnie od języka, wszystkie wersje systemu Windows zawierają te same funkcje. Zamiast osobnych wersji językowych system korzysta z pojedynczych danych binarnych o zasięgu globalnym. Dzięki temu jedna instalacja może obsługiwać wiele języków (przez dodawanie różnych pakietów językowych). Aplikacje mogą również skorzystać z funkcji systemu Windows, które zezwalają na pojedyncze, globalne dane binarne aplikacji, które mogą obsługiwać wiele języków.

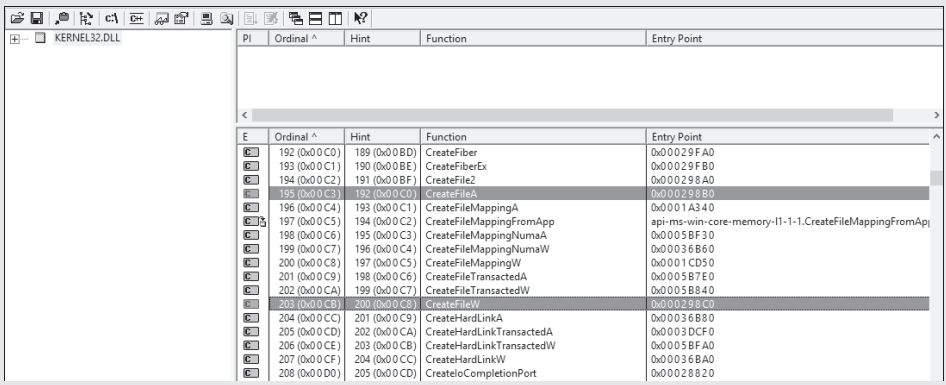
Uwaga. Starsze systemy operacyjne Windows 9*x* były pozbawione natywnej obsługi standardu Unicode. Był to jeszcze jeden powód ku temu, aby utworzyć dwie funkcje na potrzeby standardów Unicode i ANSI. Na przykład funkcja API `CreateFile` systemu Windows wcale nie jest funkcją. Jest to makro rozwijane do postaci jednej z dwóch funkcji: `CreateFileA` (ANSI) lub `CreateFileW` (Unicode, gdzie *W* odpowiada słowu *wide*). Rozwinięcie oparte jest na stałej kompilacji o nazwie `UNICODE`. Definiowana jest ona domyślnie w projektach C++ środowiska Visual Studio, gdyż korzystniejsze jest używanie funkcji Unicode. W miejsce odpowiedniego makra można jednak zastosować jawnie podaną nazwę funkcji. W zamieszczonym poniżej eksperymencie zaprezentowano takie pary funkcji.

EKSPERYMENT: Wyświetlanie wyeksportowanych funkcji

W tym eksperymencie zostanie użyte narzędzie Dependency Walker do wyświetlenia funkcji wyeksportowanych z biblioteki DLL podsystemu systemu Windows.

1. Pod adresem *http://www.dependencywalker.com/* pobierz narzędzie Dependency Walker. Jeśli używasz systemu 32-bitowego, pobierz 32-bitową wersję narzędzia. W przypadku systemu 64-bitowego pobierz wersję 64-bitową. Wyodrębnij następnie pobrany plik ZIP do wybranego folderu.
2. Uruchom narzędzie (plik *depends.exe*). Wyświetl następnie menu *File* i wybierz pozycję *Open*, aby przejść do folderu *C:\Windows\System32* (zakładając, że system Windows zainstalowano na dysku *C*). Zlokalizuj plik *kernel32.dll* i kliknij przycisk *Otwórz*.
3. Narzędzie Dependency Walker może wyświetlić okno komunikatu ostrzeżenia. Zignoruj i zamknij okno komunikatu.
4. Widocznych będzie kilka widoków z poziomymi i pionowymi paskami podziału. Upewnij się, że pozycja wybrana w lewym górnym widoku drzewa to plik *kernel32.dll*.

5. Spójrz na drugi widok od góry po prawej stronie. Widok ten wyszczególnia wyeksportowane funkcje dostępne w pliku *kernel32.dll*. Kliknij nagłówek listy *Function*, aby zastosować sortowanie według nazwy. Znajdź następnie funkcję `CreateFileA`. Niewiele niżej znajdziesz funkcję `CreateFileW`.



6. Jak widać, większość funkcji zawierających przynajmniej jeden argument typu łańcuchowego to w rzeczywistości pary funkcji. Na powyższym zrzucie ekranu widoczne są następujące funkcje: `CreateFileMappingA`/`CreateFileMappingW`, `CreateFileTransactedA`/`CreateFileTransactedW` i `CreateFileMappingNumaA`/`CreateFileMappingNumaW`.
7. W celu zlokalizowania innych funkcji możesz przewinąć listę. Możesz też otworzyć inne pliki systemowe, takie jak *user32.dll* i *advapi32.dll*.

Uwaga. Interfejsy API oparte na modelu COM w systemie Windows używają zwykle łańcuchów Unicode, które czasami są podawane przy użyciu typu `BSTR`. Zasadniczo jest to zakończona znakiem sterującym `NULL` tablica znaków Unicode o długości łańcucha wyrażonej w bajtach, która zapisywana jest 4 bajty przed początkiem tablicy znaków w pamięci. Interfejsy API środowiska Windows Runtime korzystają wyłącznie z łańcuchów Unicode podawanych przy użyciu typu `HSTRING`, który reprezentuje niezmienną tablicę znaków Unicode.

Więcej informacji o standardzie Unicode znajdziesz pod adresem *http://www.unicode.org/* oraz w dokumentacji dotyczącej programowania w bibliotece MSDN Library.

Analizowanie wewnętrznych mechanizmów systemu Windows

Choć spora część informacji zawartych w książce jest wynikiem analizowania kodu źródłowego systemu Windows i dyskusji prowadzonych z projektantami, nie musisz *wszystkiego* przyjmować na wiarę. Wiele szczegółów dotyczących wewnętrznych mechanizmów systemu Windows może zostać uzyskanych i zademonstrowanych przy użyciu różnych dostępnych narzędzi, takich jak dołączone do systemu Windows oraz jego narzędzi debugowania. Te pakiety narzędzi omówiono w skrócie w dalszej części rozdziału.

Aby zachęcić do eksplorowania wewnętrznych mechanizmów systemu Windows, w książce rozmieszczono ramki eksperymentów opisujących kroki, jakie możesz wykonać w celu sprawdzenia określonego aspektu związanego z wewnętrznym działaniem systemu Windows (wcześniej w rozdziale zamieszczono już kilka takich ramek). Zachęcamy do spróbowania przeprowadzenia tych eksperymentów, aby w praktyce sprawdzić wiele z opisanych w książce zagadnień związanych z wewnętrznymi mechanizmami systemu Windows.

W tabeli 1.4 zamieszczono listę podstawowych narzędzi używanych w książce, a także podano, skąd je uzyskać.

TABELA 1.4. *Narzędzia służące do sprawdzania wewnętrznych mechanizmów systemu Windows*

Narzędzie	Nazwa obrazu	Źródło
Startup Programs Viewer	*AUTORUNS*	Sysinternals
Access Check	*ACCESSCHK*	Sysinternals
Dependency Walker	*DEPENDS*	*www.dependencywalker.com*
Global Flags	*GFLAGS*	Debugging Tools
Handle Viewer	*HANDLE*	Sysinternals
Kernel debuggers	*WINDBG, KD*	WDK, Windows SDK
Object Viewer	*WINOBJ*	Sysinternals
Monitor wydajności	*PERFMON.MSC*	Wbudowane narzędzie systemu Windows
Pool Monitor	*POOLMON*	WDK
Process Explorer	*PROCEXP*	Sysinternals
Process Monitor	*PROCMON*	Sysinternals
Task (Process) List	*TLIST*	Debugging Tools
Menedżer zadań	*TASKMGR*	Wbudowane narzędzie systemu Windows

Monitor wydajności i Monitor zasobów

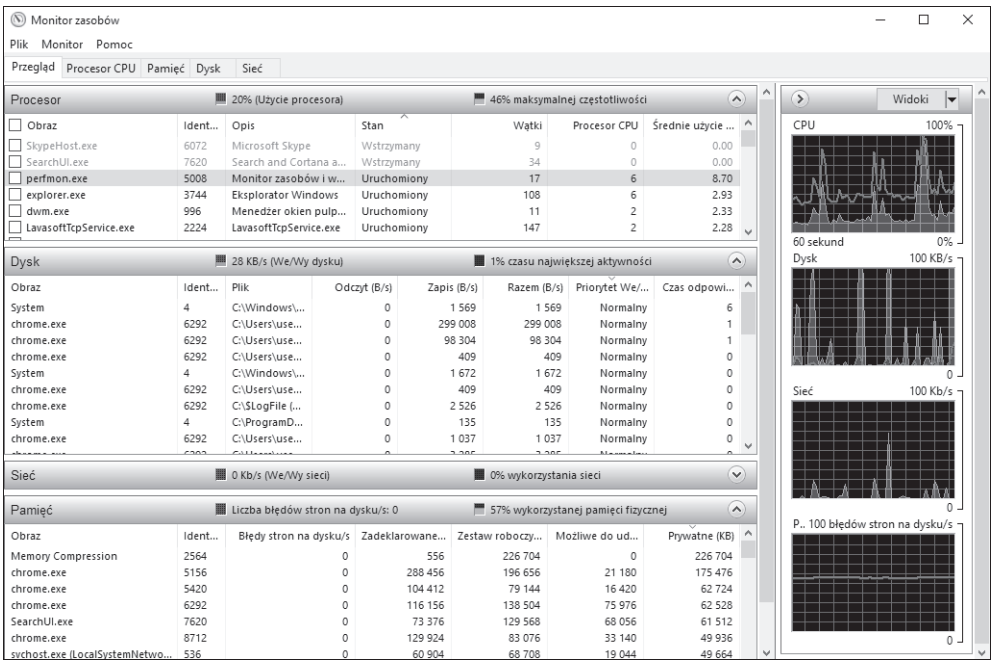
W książce odwołujemy się do narzędzia Monitor wydajności, dostępnego z poziomu folderu *Narzędzia administracyjne* panelu sterowania lub po wpisaniu polecenia **`perfmon`** w oknie dialogowym *Uruchamianie*. Dokładniej rzecz ujmując, koncentrujemy się na narzędziach Monitor wydajności i Monitor zasobów.

Uwaga. Monitor wydajności pełni trzy funkcje: monitorowanie systemu, wyświetlanie dzienników liczników wydajności i ustawianie alertów (przy użyciu zestawów modułów zbierających dane, które również zawierają dzienniki liczników wydajności oraz dane konfiguracji i śledzenia). Dla uproszczenia, gdy odwołujemy się do Monitora wydajności, mamy na myśli wchodzącą w skład tego narzędzia funkcję monitorowania systemu.

Monitor wydajności zapewnia więcej informacji o tym, jak działa system, niż jakiekolwiek inne pojedyncze narzędzie. Monitor wydajności uwzględnia setki podstawowych i rozszerzalnych liczników różnych obiektów. W wypadku każdego głównego zagadnienia opisanego w książce

dołączono tabelę z odpowiednimi licznikami wydajności systemu Windows. Monitor wydajności zawiera krótki opis każdego licznika. Aby wyświetlić opisy, wybierz licznik w oknie *Dodawanie liczników* i zaznacz opcję *Pokaż opis*.

Choć wszystkie zaprezentowane w książce operacje niskopoziomowego monitorowania systemu mogą zostać zrealizowane za pomocą Monitora wydajności, system Windows oferuje również narzędzie Monitor zasobów (dostępne z menu *Start* lub na karcie *Wydajność* menedżera zadań), które zapewnia cztery podstawowe zasoby systemowe: procesor, dysk, sieć i pamięć. W wypadku ich podstawowych stanów dla zasobów tych wyświetlany jest ten sam poziom informacji, jaki możesz znaleźć w menedżerze zadań. Monitor zasobów udostępnia też jednak sekcje, które mogą być rozwijane w celu uzyskania dodatkowych informacji. Poniżej pokazano typowy widok okna narzędzia Monitor zasobów.



Po rozwinięciu karta *Procesor CPU* wyświetla informacje dotyczące użycia procesora przez poszczególne procesy, tak jak w wypadku menedżera zadań. W obrębie tej karty dodano jednak kolumnę informującą o średnim użyciu procesora, co może pozwolić lepiej zorientować się w tym, jakie procesy są najbardziej aktywne. Karta uwzględnia również osobną sekcję prezentującą usługi oraz powiązane z nimi użycie procesora i średnie użycie. Każdy proces udostępniający usługi identyfikowany jest przez grupę usług przez niego oferowanych. Podobnie jak w wypadku narzędzia Process Explorer wybranie procesu (przez kliknięcie powiązanego z nim pola wyboru) spowoduje wyświetlenie listy nazwanych dojść otwartych przez proces, a także listy modułów (np. bibliotek DLL), które są ładowane w obszarze adresów procesu. Pole *Wyszukaj dojścia* może posłużyć do sprawdzenia, jakie procesy otwarły dojście do danego nazwanego zasobu.

Choć karta *Pamięć* wyświetla sporo tych samych informacji, jakie można uzyskać w menedżerze zadań, są one w uporządkowany sposób przedstawione dla całego systemu. Wykres słupkowy pamięci fizycznej prezentuje bieżącą organizację pamięci fizycznej podzieloną na następujące kategorie: sprzętowa zarezerwowana, w użyciu, zmodyfikowana, wstrzymana i wolna.
W rozdziale 5. znajdziesz dokładne znaczenie tych terminów.

Z kolei karta *Dysk* oferuje dla poszczególnych plików informacje związane z operacjami wejścia-wyjścia w sposób, który ułatwia identyfikację w systemie najczęściej używanych, najczęściej zapisywanych lub najczęściej odczytywanych plików. Uzyskane wyniki mogą być dalej odfiltrowywane według procesu.

Karta *Sieć* wyświetla aktywne połączenia sieciowe, procesy, do których one należą, a także informację o tym, ile danych przesyłanych jest z wykorzystaniem tych połączeń. Informacje te umożliwiają sprawdzenie aktywności sieciowej w tle, która może być trudna do określenia w inny sposób. Oprócz tego, karta zawiera aktywne w systemie połączenia TCP zorganizowane według procesów z uwzględnieniem takich danych, jak zdalny i lokalny port oraz adres, a także opóźnienie pakietów. I wreszcie, karta udostępnia listę portów nasłuchujących według procesu. Dzięki temu administrator może stwierdzić, jakie usługi lub aplikacje oczekują aktualnie na połączenia na danym porcie. Wyświetlana jest też zasada dotycząca protokołu i zapory sieciowej dla każdego portu i procesu.

Uwaga. Wszystkie liczniki wydajności systemu Windows są dostępne w sposób programowy. Więcej informacji znajdziesz w dokumentacji witryny MSDN po wpisaniu w polu wyszukiwania terminu **`liczniki wydajności`**.

Debugowanie jądra

Debugowanie jądra oznacza sprawdzanie wewnętrznych struktur danych jądra i/lub krokowe wykonywanie funkcji w jądrze. Jest to przydatny sposób kontrolowania wewnętrznych mechanizmów systemu Windows, ponieważ dzięki temu możesz uzyskać związane z nimi informacje, które nie są dostępne w wypadku żadnych innych narzędzi. Poza tym, bardziej przejrzyste staje się to, jak kod wykonywany jest wewnątrz jądra. Przed opisaniem różnych możliwych sposobów debugowania jądra przyjrzyjmy się zestawowi plików, które będą niezbędne do przeprowadzenia dowolnego typu debugowania jądra.

Symbole powiązane z debugowaniem jądra

Pliki symboli zawierają nazwy funkcji i zmiennych oraz układ i format struktur danych. Pliki są generowane przez konsolidator i używane przez debuggery do przywoływania i wyświetlania tych nazw w trakcie sesji debugowania. Informacje te nie są zwykle przechowywane w obrazie binarnym, ponieważ nie jest wymagane wykonywanie kodu. Oznacza to, że binaria są mniejsze i szybsze. Oznacza to też jednak, że podczas debugowania musisz upewnić się, że debugger może uzyskać dostęp do plików symboli powiązanych z obrazami przywoływanymi podczas sesji debugowania.

W celu użycia dowolnego z narzędzi debugowania jądra do sprawdzenia wewnętrznych struktur danych jądra systemu Windows, takich jak lista procesów, bloki wątków, lista załadowanych sterowników, informacje o użyciu pamięci itd., musisz dysponować poprawnymi plikami symboli przynajmniej dla obrazu jądra *Ntoskrnl.exe* (więcej o tym pliku dowiesz się w podrozdziale „Opis architektury systemu” rozdziału 2.). Pliki w tabeli symboli muszą być zgodne z wersją obrazu, z którego zostały uzyskane. Jeśli na przykład instalujesz pakiet Service Pack lub poprawkę systemu Windows, które aktualizują jądro, musisz uzyskać dopasowane, zaktualizowane pliki symboli.

Choć możliwe jest pobranie i zainstalowanie symboli dla różnych wersji systemu Windows, nie zawsze dostępne są zaktualizowane symbole dla poprawek. Najprostszym sposobem uzyskania poprawnej wersji symboli na potrzeby debugowania jest zastosowanie serwera symboli na żądanie firmy Microsoft za pomocą specjalnej składni ścieżki symboli podawanej w debuggerze. Na przykład następująca ścieżka symboli powoduje załadowanie przez narzędzia debugowania wymaganych symboli z internetowego serwera symboli i utrzymywanie kopii lokalnej w folderze *C:\symbols*:

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

Debugging Tools for Windows

Zestaw Debugging Tools for Windows zawiera zaawansowane narzędzia debugowania, które w książce są wykorzystywane do eksplorowania wewnętrznych mechanizmów systemu Windows. Najnowsza wersja zestawu została dołączona jako część pakietu Windows SDK (pod adresem *https://msdn.microsoft.com/en-us/library/windows/hardware/ff551063.aspx* znajdziesz więcej szczegółów o różnych typach instalacji). Narzędzia te mogą posłużyć do debugowania procesów trybu użytkownika, a także jądra.

W skład narzędzi wchodzą cztery debuggery: cdb, ntsd, kd i WinDbg. Wszystkie bazują na jednym mechanizmie debugowania, zawartym w pliku *DbgEng.dll*, który dość dobrze udokumentowano w pliku pomocy narzędzi. Oto krótki przegląd debuggerów:

- cdb i ntsd to debuggery trybu użytkownika oparte na interfejsie użytkownika konsoli. Jedyną różnicą między nimi jest to, że debugger ntsd otwiera nowe okno konsoli w wypadku aktywowania go z poziomu istniejącego okna konsoli, natomiast debugger cdb tego nie robi.
- kd to debugger trybu jądra oparty na interfejsie użytkownika konsoli.
- WinDbg może pełnić rolę debuggera zarówno trybu użytkownika, jak i trybu jądra, ale nie jednocześnie. Debugger zapewnia użytkownikowi interfejs graficzny.
- Debuggery trybu użytkownika (cdb, ntsd oraz odpowiednio uruchomione narzędzie WinDbg) są w zasadzie równorzędne. To, który z nich zostanie użyty, jest kwestią preferencji.
- Debuggery trybu jądra (kd oraz odpowiednio uruchomione narzędzie WinDbg) są w zasadzie równorzędne.

Debugowanie w trybie użytkownika

Narzędzia debugowania mogą też zostać wykorzystane do dołączenia do procesu trybu użytkownika oraz do sprawdzenia i/lub wprowadzenia zmiany w pamięci procesu. W wypadku dołączania do procesu dostępne są następujące dwie opcje:

- **Inwazyjna**. Jeśli nie określono inaczej, w momencie dołączania do działającego procesu używana jest funkcja `DebugActiveProcess` systemu Windows w celu ustanowienia połączenia między debuggerem i celem debugowania. Umożliwia to sprawdzenie i/lub wprowadzenie zmiany w pamięci procesu, ustawienie punktów wstrzymania oraz wykonanie innych funkcji debugowania. System Windows pozwala na zatrzymanie debugowania bez kończenia działania procesu docelowego, pod warunkiem że debugger zostanie odłączony, a nie wyłączony.

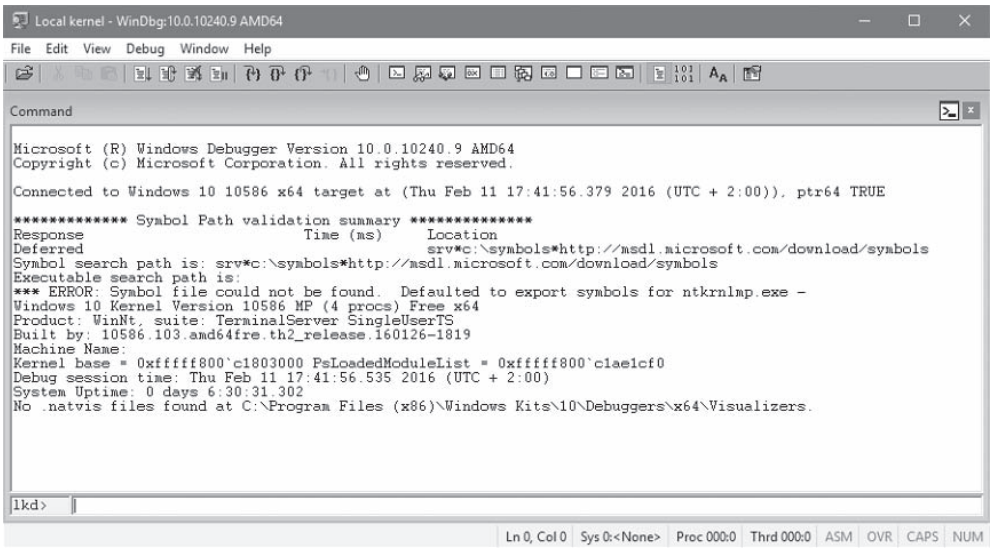
- **Nieinwazyjna**. W przypadku tej opcji debugger po prostu otwiera proces za pomocą funkcji `OpenProcess`. Debugger nie wykonuje operacji dołączenia do procesu. Umożliwia to sprawdzenie i/lub wprowadzenie zmiany w pamięci procesu docelowego, lecz nie ustawienie punktów wstrzymania. Oznacza to również, że możliwe jest nieinwazyjne dołączenie nawet wtedy, gdy w sposób inwazyjny został dołączony inny debugger.

Za pomocą narzędzi debugowania możesz też otwierać pliki zrzutów pamięci procesu w trybie użytkownika.

Debugowanie w trybie jądra

Jak już wspomniano, dostępne są dwa debuggery, które mogą posłużyć do debugowania jądra: debugger działający w trybie wiersza poleceń (*Kd.exe*) i debugger z graficznym interfejsem użytkownika (*Windbg.exe*). Możliwe jest wykonanie za pomocą tych narzędzi trzech następujących typów operacji debugowania jądra:

- Otwarcie pliku zrzutu pamięci będącego wynikiem awarii systemu Windows.
- Nawiązywanie połączenia z działającym systemem i sprawdzanie jego stanu (lub ustawianie punktów wstrzymania, jeśli debugowany jest kod sterownika urządzenia). Operacja ta wymaga dwóch komputerów: docelowego (z debugowanym systemem) i hosta (z uruchomionym debuggerem). Komputer docelowy może zostać połączony z hostem za pośrednictwem kabla Null Modem, kabla IEEE 1394, kabla do debugowania USB 2.0/3.0 lub sieci lokalnej. Rozruch systemu komputera docelowego musi się odbyć w trybie debugowania. System może zostać skonfigurowany pod kątem rozruchu w trybie debugowania za pomocą programu *Bcdedit.exe* lub *Msconig.exe* (zauważ, że może być konieczne wyłączenie opcji bezpiecznego rozruchu w ustawieniach UEFI BIOS-u). Możliwe jest też nawiązanie połączenia z wykorzystaniem nazwanego potoku (co jest przydatne podczas debugowania systemu Windows 7 lub wcześniejszych wersji systemu Windows przy użyciu oprogramowania maszyn wirtualnych, takich jak Hyper-V, Virtual Box lub VMWare Workstation) przez udostępnienie portu szeregowego systemu operacyjnego „gościa” jako urządzenia nazwanego potoku. W wypadku „gości” z systemem Windows 8 lub nowszym należy użyć debugowania opartego na sieci lokalnej, udostępniając sieć tylko z hostem za pomocą wirtualnej karty sieciowej w systemie operacyjnym „gościa”. Spowoduje to tysiąckrotny wzrost wydajności.
- System Windows umożliwia też nawiązanie połączenia z komputerem lokalnym i sprawdzenie stanu systemu. Jest to określane mianem *lokalnego debugowania jądra*. Aby je zainicjować za pomocą debuggera WinDbg, upewnij się najpierw, że system został ustawiony w trybie debugowania (na przykład przez uruchomienie programu msconig.exe, kliknięcie karty *Rozruch*, a następnie kliknięcie przycisku *Opcje zaawansowane* i zaznaczenie opcji *Debuguj* oraz zrestartowanie systemu Windows). Wywołaj debugger WinDbg z uprawnieniami administratora, wyświetl menu *File*, wybierz pozycję *Kernel Debug*, kliknij kartę *Local*, a następnie przycisk *OK* (lub użyj programu *bcdedit.exe*). Na rysunku 1.6 pokazano przykładowy ekran danych wynikowych uzyskany w 64-bitowym systemie Windows. Niektóre polecenia debuggera jądra, takie jak ustawiające punkty wstrzymania lub tworzące zrzut pamięci za pomocą polecenia `.dump`, nie zadziałają, gdy zostaną użyte w trybie lokalnego debugowania jądra. Zrzut może być jednak uzyskany przy użyciu narzędzia LiveKd, opisanego w dalszej części rozdziału.



RYSUNEK 1.6. *Lokalne debugowanie jądra*

Po nawiązaniu połączenia w trybie debugowania jądra możesz zastosować jedno z wielu poleceń rozszerzających debugger, które są też nazywane poleceniami *bang*. Są to polecenia o nazwie zaczynającej się znakiem wykrzyknika (!). Polecenia umożliwiają wyświetlenie zawartości wewnętrznych struktur danych, takich jak wątki, procesy, pakiety żądań wejścia-wyjścia, oraz informacji dotyczących zarządzania pamięcią. W książce uwzględniono odpowiednie polecenia debuggera jądra i jego dane wyjściowe w miejscach, w których mają one zastosowanie odnośnie poszczególnych omawianych zagadnień. Znakomitym dodatkowym zasobem jest plik pomocy *Debugger.chm*, znajdujący się w folderze instalacyjnym debuggera WinDbg. W pliku tym udokumentowano wszystkie funkcje i rozszerzenia debuggera jądra. Ponadto polecenie `dt` (*display type*) może sformatować więcej niż 1000 struktur jądra, ponieważ pliki symboli jądra dla systemu Windows zawierają informacje o typach, których debugger może użyć do formatowania struktur.

EKSPERYMENT: Wyświetlanie informacji o typach powiązanych ze strukturami jądra

Aby wyświetlić listę struktur jądra, których informacje o typach uwzględniono w symbolach jądra, w oknie debuggera jądra wpisz polecenie **`dt nt!_*`**. Poniżej zaprezentowano część przykładowych danych wynikowych (*ntkrnlmp* to nazwa pliku wewnętrznego jądra 64-bitowego; więcej szczegółów podano w rozdziale 2.).

```
lkd> dt nt!_*
          ntkrnlmp!_KSYSTEM_TIME
          ntkrnlmp!_NT_PRODUCT_TYPE
          ntkrnlmp!_ALTERNATIVE_ARCHITECTURE_TYPE
          ntkrnlmp!_KUSER_SHARED_DATA
          ntkrnlmp!_ULARGE_INTEGER
          ntkrnlmp!_TP_POOL
          ntkrnlmp!_TP_CLEANUP_GROUP
```

```
ntkrnlmp!_ACTIVATION_CONTEXT
          ntkrnlmp!_TP_CALLBACK_INSTANCE
          ntkrnlmp!_TP_CALLBACK_PRIORITY
          ntkrnlmp!_TP_CALLBACK_ENVIRON_V3
          ntkrnlmp!_TEB
```

Możesz również zastosować polecenie `dt` do wyszukania konkretnych struktur, używając jego funkcji wyszukiwania za pomocą znaków wieloznacznych. Jeśli na przykład byłaby szukana nazwa struktury obiektu przerwania, należałoby wpisać polecenie **`dt nt!_*interrupt*`**:

```
lkd> dt nt!_*interrupt*
          ntkrnlmp!_KINTERRUPT_MODE
          ntkrnlmp!_KINTERRUPT_POLARITY
          ntkrnlmp!_PEP_ACPI_INTERRUPT_RESOURCE
          ntkrnlmp!_KINTERRUPT
          ntkrnlmp!_UNEXPECTED_INTERRUPT
          ntkrnlmp!_INTERRUPT_CONNECTION_DATA
          ntkrnlmp!_INTERRUPT_VECTOR_DATA
          ntkrnlmp!_INTERRUPT_HT_INTR_INFO
          ntkrnlmp!_INTERRUPT_REMAPPING_INFO
```

Możesz następnie użyć polecenia `dt` w celu sformatowania określonej struktury w następujący sposób (w wypadku struktur debugger nie rozróżnia wielkości liter):

```
lkd> dt nt!_KINTERRUPT
   +0x000 Type             : Int2B
   +0x002 Size             : Int2B
   +0x008 InterruptListEntry : _LIST_ENTRY
   +0x018 ServiceRoutine   : Ptr64     unsigned char
   +0x020 MessageServiceRoutine : Ptr64     unsigned char
   +0x028 MessageIndex     : Uint4B
   +0x030 ServiceContext   : Ptr64 Void
   +0x038 SpinLock         : Uint8B
   +0x040 TickCount        : Uint4B
   +0x048 ActualLock       : Ptr64 Uint8B
   +0x050 DispatchAddress  : Ptr64     void
   +0x058 Vector           : Uint4B
   +0x05c Irql             : UChar
   +0x05d SynchronizeIrql  : UChar
   +0x05e FloatingSave     : UChar
   +0x05f Connected        : UChar
   +0x060 Number           : Uint4B
   +0x064 ShareVector      : UChar
   +0x065 EmulateActiveBoth : UChar
   +0x066 ActiveCount      : Uint2B
   +0x068 InternalState    : Int4B
   +0x06c Mode             : _KINTERRUPT_MODE
   +0x070 Polarity         : _KINTERRUPT_POLARITY
   +0x074 ServiceCount     : Uint4B
   +0x078 DispatchCount    : Uint4B
   +0x080 PassiveEvent     : Ptr64 _KEVENT
   +0x088 TrapFrame        : Ptr64 _KTRAP_FRAME
   +0x090 DisconnectData   : Ptr64 Void
   +0x098 ServiceThread    : Ptr64 _KTHREAD
   +0x0a0 ConnectionData   : Ptr64 _INTERRUPT_CONNECTION_DATA
   +0x0a8 IntTrackEntry    : Ptr64 Void
   +0x0b0 IsrDpcStats      : _ISRDPCSTATS
   +0x0f0 RedirectObject   : Ptr64 Void
   +0x0f8 Padding          : [8] UChar
```

Zauważ, że domyślnie polecenie `dt` nie pokazuje podstruktur (struktury w obrębie innych struktur). Aby je wyświetlić, użyj opcji `-r` lub `-b`. Zastosowanie jednej z tych opcji na przykład do wyświetlenia obiektu przerwania jądra spowoduje zaprezentowanie formatu struktury `_LIST_ENTRY` przechowywanej w polu `InterruptListEntry` (w dokumentacji opisano dokładne różnice między opcjami `-r` i `-b`).

```
lkd> dt nt!_KINTERRUPT -r
   +0x000 Type             : Int2B
   +0x002 Size             : Int2B
   +0x008 InterruptListEntry : _LIST_ENTRY
      +0x000 Flink            : Ptr64 _LIST_ENTRY
         +0x000 Flink            : Ptr64 _LIST_ENTRY
         +0x008 Blink            : Ptr64 _LIST_ENTRY
      +0x008 Blink            : Ptr64 _LIST_ENTRY
         +0x000 Flink            : Ptr64 _LIST_ENTRY
         +0x008 Blink            : Ptr64 _LIST_ENTRY
   +0x018 ServiceRoutine   : Ptr64     unsigned char
```

Polecenie `dt` pozwala nawet określić poziom rekurencji struktur przez dodanie liczby do opcji `-r`. W następującym przykładzie użyto jednego poziomu rekurencji:

```
lkd> dt nt!_KINTERRUPT -r1
```

W pliku pomocy zestawu Debugging Tools for Windows wyjaśniono, jak konfigurować i stosować debuggery jądra. W dokumentacji zestawu WDK zamieszczono dodatkowe szczegóły dotyczące korzystania z debuggerów jądra, które przeznaczone są głównie dla osób tworzących sterowniki urządzeń.

Narzędzie LiveKd

LiveKd to darmowe narzędzie dostępne w witrynie Sysinternals, które umożliwia użycie standardowych debuggerów jądra firmy Microsoft, które właśnie opisano, do sprawdzania działającego systemu bez konieczności przeprowadzania jego rozruchu w trybie debugowania. Takie rozwiązanie może być przydatne, gdy niezbędne okaże się rozwiązywanie problemów na poziomie jądra w wypadku komputera, którego nie uruchomiono w trybie debugowania. Określone problemy mogą być trudne do właściwego odtworzenia, dlatego rozruch z włączoną opcją *debugowania* może nie umożliwić ujawnienia błędu w prosty sposób.

Narzędzie LiveKd jest uruchamiane tak jak w wypadku debuggera WinDbg lub kd. Narzędzie to przekazuje wybranemu debuggerowi wszystkie podane opcje wiersza poleceń. Domyślnie narzędzie uruchamia debugger jądra trybu wiersza poleceń (kd). Aby załadować debugger WinDbg, użyj opcji `-w`. W celu wyświetlenia plików pomocy dla opcji narzędzia LiveKd zastosuj opcję `-?`.

Narzędzie LiveKd prezentuje debuggerowi plik symulowanego zrzutu pamięci po awarii, dlatego możesz wykonać dowolne operacje, które są obsługiwane w przypadku zrzutu pamięci po awarii. Ponieważ narzędzie to na potrzeby symulowanego zrzutu korzysta z pamięci fizycznej, debugger jądra może mieć do czynienia z sytuacjami, w których struktury danych są w trakcie wprowadzania zmian przez system, a tym samym są niespójne. Każdorazowo po wywołaniu debugger zaczyna działać z wykorzystaniem nowego widoku stanu systemu. Aby odświeżyć migawkę, wprowadź polecenie `q` w celu zakończenia pracy debuggera. Narzędzie LiveKd zapyta, czy zamierzasz ponownie uruchomić debugger. Jeśli debugger zapętli się podczas wyświetlania danych wynikowych, użyj kombinacji klawiszy *Ctrl+C*, aby przerwać tę operację i zakończyć działanie debuggera. W przypadku jego zawieszenia użyj kombinacji klawiszy *Ctrl+Break*, która spowoduje zakończenie procesu debuggera. Narzędzie LiveKd zapyta następnie, czy debugger ma zostać ponownie uruchomiony.

Windows Software Development Kit

Zestaw Windows Software Development Kit (SDK) stanowi część programu subskrypcji MSDN. Możesz go pobrać bezpłatnie pod adresem *https://developer.microsoft.com/en-US/windows/downloads/windows-10-sdk*. W trakcie instalacji środowisko Visual Studio również zapewnia opcję instalacji zestawu SDK. Wersje zawarte w zestawie Windows SDK zawsze są zgodne z najnowszą wersją systemu operacyjnego Windows, natomiast wersja dołączona do środowiska Visual Studio może być starszą wersją, która była aktualna w chwili jej udostępnienia. Oprócz zestawu Debugging Tools for Windows, zestaw SDK zawiera pliki nagłówkowe języka C oraz biblioteki niezbędne do kompilowania i konsolidowania aplikacji dla systemu Windows. Z perspektywy wewnętrznych mechanizmów systemu Windows godne zainteresowania elementy zestawu Windows SDK obejmują pliki nagłówkowe interfejsu API systemu Windows (na przykład zawarte w folderze *C:\Program Files (x86)\Windows Kits\10\Include*) oraz narzędzia SDK (poszukaj folderu *Bin*). Godna uwagi jest też dokumentacja. Jest ona dostępna w trybie online; można ją też pobrać w celu korzystania w trybie offline. Kilka spośród tych narzędzi jest też dołączonych w postaci przykładowego kodu źródłowego zarówno w zestawie Windows SDK, jak i bibliotece MSDN Library.

Windows Driver Kit

Zestaw Windows Driver Kit (WDK) również jest dostępny w ramach programu subskrypcji MSDN. Tak jak zestaw Windows SDK, zestaw WDK może zostać bezpłatnie pobrany. Jego dokumentację dołączono do biblioteki MSDN Library.

Choć zestaw WDK przeznaczony jest dla projektantów sterowników urządzeń, stanowi on bogate źródło informacji o wewnętrznych mechanizmach systemu Windows. Na przykład w rozdziale 6. opisano architekturę systemu wejścia-wyjścia, model sterowników oraz podstawowe struktury danych sterowników urządzeń, ale nie omówiono w nim szczegółowo poszczególnych funkcji obsługi jądra. Dokumentacja zestawu WDK zawiera obszerny opis, zarówno w postaci kursu, jak i materiału referencyjnego, wszystkich funkcji obsługi jądra i mechanizmów systemu Windows używanych przez sterowniki urządzeń.

Oprócz dokumentacji, zestaw WDK uwzględnia pliki nagłówkowe (w szczególności pliki *ntddk.h*, *ntifs.h* i *wdm.h*), które definiują kluczowe wewnętrzne struktury danych i stałe, a także interfejsy do wielu wewnętrznych programów systemowych. Pliki te przydają się podczas eksplorowania wewnętrznych struktur danych systemu Windows za pomocą debuggera jądra, ponieważ choć ogólny układ i zawartość tych struktur zaprezentowano w książce, nie uwzględniono szczegółowych opisów na poziomie pól (na przykład dotyczących wielkości i typów danych). Kilka tych struktur danych, takich jak nagłówki rozdzielacza obiektów, bloki oczekiwania, zdarzenia, muteksy, semafory itp., opisano jednak szczegółowo w zestawie WDK.

Aby zaznajomić się z systemem operacji wejścia-wyjścia i modelem sterowników w zakresie szerszym niż zaprezentowany w książce, przeczytaj dokumentację zestawu WDK, a zwłaszcza podręczniki *Kernel-Mode Driver Architecture Design Guide* i *Kernel-Mode Driver Reference*. Za wartościową możesz też uznać książkę *Programming the Microsoft Windows Driver Model* (wydanie drugie; Microsoft Press, 2002 r.) autorstwa Waltera Oneya oraz książkę *Developing Drivers with the Windows Driver Foundation* napisaną przez Penny Orwick i Guya Smitha (Microsoft Press, 2007 r.).

Narzędzia z witryny Sysinternals

W wielu eksperymentach zamieszczonych w książce wykorzystano darmowe narzędzia, które możesz pobrać z witryny Sysinternals. Większość tych narzędzi napisał Mark Russinovich, współautor książki. Do najpopularniejszych narzędzi należą Process Explorer i Process Monitor. Zauważ, że wiele z tych narzędzi uwzględnia instalację i uruchamianie sterowników urządzeń trybu jądra, dlatego wymagają uprawnień administratora lub zwiększonych uprawnień. Niemniej jednak w wypadku standardowego konta użytkownika lub mającego mniejsze uprawnienia część narzędzi może działać z ograniczoną funkcjonalnością.

Ponieważ narzędzia z witryny Sysinternals są często aktualizowane, upewnij się, że dysponujesz najnowszą wersją. Aby być powiadamianym o aktualizacjach narzędzi, możesz śledzić blog witryny Sysinternals, który oferuje kanał RSS. W celu zaznajomienia się z opisem wszystkich narzędzi, sposobem korzystania z nich oraz analizami przypadków rozwiązywania problemów zajrzyj do książki *Windows Sysinternals Administrator's Reference* autorstwa Marka Russinovicha i Aarona Margosisa (Microsoft Press, 2011 r.). Do zadawania pytań i prowadzenia dyskusji dotyczących narzędzi użyj forów witryny Sysinternals.

Podsumowanie

W rozdziale dokonano wprowadzenia do kluczowych zagadnień technicznych i terminów związanych z systemem Windows, które będą używane w książce. Zaprezentowano też w skrócie wiele przydatnych narzędzi służących do poznawania wewnętrznych mechanizmów systemu Windows. Jesteś już gotowy do rozpoczęcia eksploracji wnętrza systemu. Na początek przyjrzymy się ogólnie architekturze systemu i jej kluczowym komponentom.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

Z

Ż

Windows
od podszewki — poznaj i zrozum!

Systemy operacyjne Windows 10 i Windows Server 2016 zdecydowanie różnią się od swoich poprzedników. Są bardziej złożone, a niektóre zastosowane rozwiązania można określić jako zaawansowane i wysublimowane. Zwłaszcza znajomość wewnętrznych mechanizmów systemu, architektury jądra i systemowego modelu bezpieczeństwa okazuje się niezwykle istotna dla programistów i inżynierów odpowiadających za bezpieczeństwo. Bez tej wiedzy trudno mówić o prawdziwie niezawodnej pracy oprogramowania tworzonego dla maszyn pracujących pod kontrolą najnowszych wersji systemu Windows.

Ta książka jest pierwszym tomem klasycznego przewodnika po wewnętrznych mechanizmach podstawowych komponentów systemu Windows. Omówiono w niej jego architekturę, sposoby implementowania i modyfikacji procesów, przetwarzania wątków oraz korzystania z pamięci fizycznej i wirtualnej. Sporo miejsca poświęcono operacjom wejścia-wyjścia oraz integracji ze sterownikami poszczególnych urządzeń. Szczegółowo przedstawiono zabezpieczenia wbudowane w system. Projektanci oprogramowania, specjaliści do spraw bezpieczeństwa oraz administratorzy systemów informatycznych znajdą tu wiele ważnych informacji, dzięki którym dogłębnie zrozumieją sposób działania systemu, co pozwoli im na podejmowanie lepszych decyzji.

Najważniejsze zagadnienia:

- wprowadzenie do wewnętrznych mechanizmów systemu Windows
- omówienie komponentów architektury systemu
- procesy, zadania, wątki
- funkcjonowanie menedżera pamięci i sposoby jej modyfikacji
- operacje wejścia-wyjścia i obsługa urządzeń peryferyjnych
- mechanizmy zabezpieczeń i zwalczanie złośliwego oprogramowania

Pavel Yosifovich specjalizuje się w technikach i narzędziach stosowanych przez Microsoft. Otrzymał tytuł Microsoft MVP. Programowanie jest jego pasją już od czasu komputerów 8-bitowych.
Alex Ionescu jest niekwestionowanym autorytetem w dziedzinie niskopoziomowego programowania systemów operacyjnych, udoskonalania zabezpieczeń i inżynierii wstecznej.
Mark E. Russinovich jest wybitnym specjalistą w dziedzinie systemów rozproszonych. W Microsofcie zajmuje się chmurową platformą Azure.
David A. Solomon jest autorem wielu książek o systemie Windows. O jądrze tego systemu operacyjnego wie chyba wszystko. W latach 1993 i 2005 otrzymał tytuł Microsoft MVP.

Microsoft

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-3901-9
9 788328 339019
Cena: 129,00 zł

INFORMATYKA W NAJLEPSZYM WYDANIU