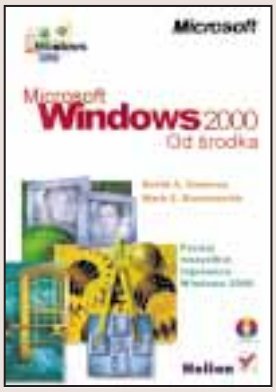
MS Windows 2000 od środka



Autorzy: David A. Solomon, Mark E. Russinovich
ISBN: 83-7197-821-9
Tytuł oryginału: Inside MS Windows 2000, 3rd Edition
Format: B5, stron: 784
Zawiera CD-ROM

Książka dostarczy Ci unikatowej wiedzy na temat wewnętrznych mechanizmów działania systemu Windows 2000. Została napisana we współpracy z zespołem tworzącym Windows 2000, a jej autorzy mieli pełen dostęp do kodu źródłowego systemu. Znajdziesz tutaj informacje niedostępne w innych źródłach, pozwalające pisać wydajniejsze aplikacje przeznaczone dla platformy Windows 2000. Z książki skorzystają również administratorzy systemowi. Zrozumienie tajemnic mechanizmów systemu operacyjnego ułatwi im odnalezienie źródła problemu w przypadku awarii.

Poznaj wszystkie tajemnice Windows 2000:

- Podstawowe pojęcia: Win32 API, procesy, wątki i prace, tryb jądra i tryb użytkownika
- Architektura systemu — kluczowe komponenty
- Mechanizmy systemowe: obsługa przerwań, menedżer obiektów, synchronizacja, systemowe wątki wykonawcze
- Proces uruchamiania i zamykania systemu
- Mechanizmy zarządzania,
- Zarządzanie procesami, wątkami i pracami
- Zarządzanie pamięcią
- Zagadnienia związane z bezpieczeństwem systemu
- Podsystem wejścia-wyjścia
- Zarządzanie pamięcią masową, systemy plików
- Menedżer pamięci podręcznej
- Praca systemu Windows 2000 w sieciach lokalnych i Internecie

Jeśli poważnie myślisz o tworzeniu oprogramowania dla Windows 2000 lub chcesz nim w pełni profesjonalnie administrować, Microsoft Windows 2000. Od środka stanowić będzie ważną pozycję w Twojej bibliotece. Bardziej szczegółowego opisu tego systemu operacyjnego nie znajdziesz w żadnej innej książce.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

2

Architektura systemu

Po omówieniu pojęć, zagadnień i narzędzi, których znajomość jest konieczna, pora przejść do założeń projektowych i struktury systemu Microsoft Windows 2000 (dawniej Windows NT). Ten rozdział prezentuje ogólną architekturę systemu — kluczowe komponenty, ich współpracę i kontekst, w jakim działają. Aby zbudować podstawy do zrozumienia wewnętrznej struktury Windows 2000, musimy omówić założenia i cele projektowe, które określono przy tworzeniu pierwszego projektu i specyfikacji systemu.

Wymagania i cele projektowe

Poniższe wymagania nadały kierunek specyfikacji systemu Windows NT w 1989 r.:

- Stworzyć w pełni 32-bitowy, wielodostępny system operacyjny, zapewniający wielowątkowość z wywłaszczeniem i pamięć wirtualną.
- Obsługiwać wiele platform i architektur sprzętowych.
- Zapewnić obsługę i skalowalność symetrycznych systemów wieloprzetwarzania.
- Stworzyć platformę przetwarzania rozproszonego, funkcjonującą zarówno jako klient, jak i jako serwer.
- Umożliwić uruchomienie większości istniejących 16-bitowych aplikacji dla MS-DOS oraz Microsoft Windows 3.1.
- Spełnić rządowe wymagania zgodności ze standardem POSIX 1003.1.
- Spełnić rządowe i przemysłowe wymagania dotyczące bezpieczeństwa systemu komputerowego.
- Zapewnić możliwość adaptacji do warunków rynku międzynarodowego poprzez obsługę standardu Unicode.

Zespół projektowy Windows NT uznał, że każda decyzja dotycząca systemu musi bazować na przyjętych celach projektowych. Są to:

- **Rozszerzalność**. Kod musi być napisany w sposób umożliwiający wygodny rozwój, postępujący za zmieniającymi się wymaganiami rynku.
- **Przenośność**. System musi obsługiwać wiele platform sprzętowych. Musi także istnieć możliwość relatywnie łatwego przenoszenia go na nowe platformy, zgodnie z trendami rynkowymi.
- **Niezawodność i stabilność**. System powinien być odporny zarówno na wewnętrzne awarie, jak i na zagrożenia zewnętrzne. Aplikacje nie mogą mieć możliwości zagrożenia systemowi operacyjnemu ani innym aplikacjom.
- **Zgodność**. Mimo ze Windows NT ma rozszerzać istniejące technologie, interfejs użytkownika oraz API powinien być zgodny ze starszymi wersjami Windows oraz z MS-DOS. Windows NT powinien również dobrze współpracować z innymi systemami, takimi jak Unix, OS/2 czy NetWare.
- **Wydajność**. W ramach pozostałych założeń projektowych system powinien zapewniać wydajność i czas reakcji na najwyższym, możliwym na danej platformie sprzętowej, poziomie.

W miarę zagłębiania się w tajniki wewnętrznej struktury i działania Windows 2000 okaże się, w jaki sposób te początkowe założenia i wymagania rynkowe zostały z powodzeniem wplecione w konstrukcję systemu. Ale przed dokładną analizą omówimy ogólny model projektowy Windows 2000 oraz jego podobieństwa i różnice z innymi współczesnymi systemami operacyjnymi.

Windows 2000 kontra Consumer Windows

Windows 2000 oraz Consumer Windows (Windows 95, Windows 98 i Windows Millenium Edition) wchodzą w skład „rodziny systemów operacyjnych Windows". Rodzina ta posiada wspólny interfejs programowy (Win32 oraz COM), a także, w niektórych przypadkach, wspólny kod. Windows 2000, Windows 98 i Windows Millenium Edition korzystają również ze wspólnego modelu sterowników urządzeń o nazwie Windows Driver Model (WDM). Model ten zostanie bardziej szczegółowo omówiony w rozdziale 9.

Od momentu rozpoczęcia prac nad Windows NT Microsoft jasno określał ten system jako strategiczną platformę przyszłości — nie tylko dla serwerów i komputerów w przedsiębiorstwach, ale także docelowo dla komputerów domowych. Poniższa lista wymienia pewne różnice w architekturze oraz zalety Windows 2000 względem Consumer Windows:

- Windows 2000 obsługuje systemy wieloprocesorowe — Consumer Windows nie.
- System plików Windows 2000 zapewnia bezpieczeństwo (poprzez na przykład uznaniową kontrolę dostępu). Nie można tego powiedzieć o systemie plików wykorzystywanym w Consumer Windows.
- Windows 2000 jest w pełni 32-bitowym systemem operacyjnym — nie zawiera kodu 16-bitowego, poza fragmentami kodu wspierającymi uruchamianie 16-bitowych aplikacji Windows. Systemy Consumer Windows zawierają dużą ilość 16-bitowego kodu odziedziczonego po przodkach: Windows 3.1 i MS-DOS.

- System Windows 2000 jest w pełni wielodostępny — czego nie można powiedzieć o znaczącej części Consumer Windows (chodzi tu głównie o 16-bitowy kod odziedziczony po Windows 3.1). Ten nie wielodostępny kod to większość funkcji związanych z grafiką i systemem zarządzania interfejsem użytkownika (GDI oraz USER). W momencie, gdy aplikacja 32-bitowa uruchomiona w Consumer Windows próbuje wywołać funkcję systemową zaimplementowaną poprzez nie wielodostępny kod 16-bitowy, musi najpierw dokonać ogólnosystemowej blokady (nazywanej *mutexem*), aby nie dopuścić innych wątków do tego kodu. Co gorsza, aplikacja 16-bitowa utrzymuje tę blokadę *podczas pracy*. W rezultacie, mimo że jądro Consumer Windows zawiera 32-bitowy mechanizm szeregowania zadań z wywłaszczeniem, aplikacje często wykonywane są jednowątkowo, ponieważ większość systemu jest zaimplementowana w postaci nie wielodostępnego kodu.
- Windows 2000 zapewnia możliwość uruchomienia 16-bitowych aplikacji Windows w ich własnej przestrzeni adresowej — systemy Consumer Windows zawsze uruchamiają 16-bitowe aplikacje we wspólnej przestrzeni adresowej, przez co mogą one przeszkadzać sobie (i zawieszać się) nawzajem.
- Wspólna przestrzeń adresowa w Windows 2000 jest widoczna jedynie dla procesów, które korzystają z tego samego segmentu pamięci współdzielonej. (W Win32 API segment pamięci współdzielonej nazywa się *obiektem odwzorowania pliku* — ang. *file mapping object*). W systemach Consumer Windows dostęp i zapisywanie w pamięci współdzielonej są możliwe dla wszystkich procesów. Tak więc każdy proces może wykonać zapis do obiektu odwzorowania pliku.
- Systemy Consumer Windows posiadają również krytyczne dla systemu strony pamięci, których modyfikacja jest możliwa w trybie użytkownika. Dzięki temu aplikacja użytkownika jest w stanie doprowadzić do zakłócenia lub załamania pracy systemu.

Jedyną rzeczą możliwą w systemach Consumer Windows, która nigdy nie będzie możliwa w Windows 2000, jest zdolność uruchamiania *wszystkich* starszych aplikacji dla MS-DOS i Windows 3.1 (zwłaszcza aplikacji wymagających bezpośredniego dostępu do sprzętu) oraz 16-bitowych sterowników urządzeń dla MS-DOS-a. Podczas gdy stuprocentowa zgodność z systemami MS-DOS i Windows 3.1 była nadrzędnym założeniem w Windows 95, założeniem w przypadku Windows NT była możliwość uruchomienia *większości* istniejących aplikacji 16-bitowych, przy zachowaniu spójności i niezawodności systemu.

Model systemu operacyjnego

W większości wielodostępnych systemów operacyjnych aplikacje są oddzielone od samego systemu — kod systemu operacyjnego jest wykonywany w uprzywilejowanym trybie procesora (w tej książce określanym jako *tryb jądra*), z dostępem do danych systemowych oraz sprzętu. Kod aplikacji jest wykonywany w pozbawionym przywilejów trybie procesora (zwanym *trybem użytkownika*), z ograniczonym dostępem do interfejsów, danych systemowych oraz bez bezpośredniego dostępu do sprzętu. Gdy program w trybie

użytkownika wywołuje usługę systemową, procesor przechwytuje to wywołanie i przełącza wywołujący wątek w tryb jądra. Gdy wykonanie usługi systemowej kończy się, system operacyjny przełącza kontekst wątku z powrotem do trybu użytkownika i kontynuuje wykonanie aplikacji.

Windows 2000 jest podobny do większości systemów uniksowych w tym sensie, że jest systemem monolitycznym — większość kodu systemu operacyjnego i sterowników urządzeń współdzieli tę samą, chronioną przestrzeń adresową trybu jądra. Oznacza to, że każdy komponent systemu lub sterownik urządzenia może potencjalnie zakłócić integralność danych wykorzystywanych przez inne komponenty systemu.

Wszystkie te komponenty systemu operacyjnego są oczywiście w pełni chronione przed działaniem błędnych aplikacji, ponieważ aplikacje nie mają bezpośredniego dostępu do kodu i danych uprzywilejowanej części systemu operacyjnego (mimo że mogą wywoływać inne usługi jądra). Ta ochrona jest powodem, dla którego Windows 2000 ma reputację systemu niezawodnego i stabilnego, zarówno w roli serwera aplikacji, jak i stacji roboczej. Ma reputację jednak szybkiego i sprawnego z punktu widzenia podstawowych usług systemu operacyjnego, jak zarządzanie pamięcią wirtualną, dostęp do plików, obsługa sieci oraz współdzielenie drukarek i plików.

Czy Windows 2000 jest systemem z mikrojądrem?

Mimo że niektórzy twierdzą inaczej, Windows 2000 nie jest systemem operacyjnym z mikrojądrem w klasycznym sensie tego określenia, oznaczającego system, w którym główne komponenty (jak zarządzanie pamięcią, procesami czy zasobami wejścia-wyjścia) są osobnymi procesami — każdy w prywatnej przestrzeni adresowej — opartymi na prostym zestawie usług zapewnianych przez mikrojądro. Na przykład system operacyjny Mach, opracowany laboratoriach Carnegie Mellon University, będący współczesnym przykładem architektury z mikrojądrem, zawiera minimalnej wielkości jądro, zapewniające szeregowanie procesów, przekazywanie komunikatów, pamięć wirtualną oraz sterowniki urządzeń. Pozostałe komponenty, włącznie z różnymi interfejsami programowymi, systemami plików i obsługą sieci, funkcjonują w trybie użytkownika. Jednakże komercyjne implementacje systemu z mikrojądrem Mach z reguły zezwalają na wykonanie w trybie jądra przynajmniej komponentów obsługujących system plików, sieci oraz zarządzanie pamięcią. Powód jest prosty: klasyczna architektura oparta na mikrojądrze jest niepraktyczna ze względów komercyjnych z powodu zbyt małej wydajności.

Czy fakt, że tak wiele komponentów Windows 2000 działa w trybie jądra, oznacza, że system ten jest bardziej podatny na awarie niż system z prawdziwym mikrojądrem? Ależ skąd. Można rozważyć następujący scenariusz: załóżmy, że kod obsługujący system plików zawiera błąd, sporadycznie powodujący awarię. W tradycyjnym systemie operacyjnym lub zmodyfikowanym systemie z mikrojądrem błąd w kodzie trybu jądra, takim jak procedury menedżera pamięci lub systemu plików, prawdopodobnie spowoduje załamanie całego systemu. W klasycznym systemie operacyjnym z mikrojądrem takie komponenty działają w trybie użytkownika, tak więc teoretycznie błąd oznacza zakończenie procesu komponentu. Ale z praktycznego punktu widzenia system padnie, ponieważ przetrwanie awarii takiego krytycznego procesu prawdopodobnie będzie niemożliwe.

Komponenty trybu jądra Windows 2000 są również zgodne z paradygmatami projektowania zorientowanego obiektowo. Na przykład nie odwołują się nawzajem do swoich wewnętrznych struktur danych. Zamiast tego korzystają z formalnych interfejsów, umożliwiających dostęp i modyfikację tych struktur danych.

Oprócz powszechnego stosowania obiektów do reprezentacji współdzielonych zasobów systemowych, Windows 2000 nie jest systemem obiektowym w ścisłym tego słowa znaczeniu. Większość kodu systemu operacyjnego napisano w języku C. Ma to uzasadnienie w przenośności oraz powszechnej dostępności narzędzi związanych z tym językiem. C nie obsługuje w bezpośredni sposób konstrukcji znanych z języków obiektowych, jak dynamiczne przyporządkowywanie typów danych, polimorfizm czy dziedziczenie klas. Dlatego oparta na języku C implementacja obiektów w Windows 2000 korzysta z cech poszczególnych języków obiektowych, ale nie jest od nich zależna.

Przenośność

System Windows 2000 został zaprojektowany z myślą o obsłudze rozmaitych architektur sprzętowych, włącznie z systemami typu CISC opartymi na procesorach firmy Intel oraz systemami typu RISC. Pierwsza wersja Windows NT obsługiwała architektury x86 oraz MIPS. Obsługa procesora Alpha AXP firmy Digital Equipment Corporation (DEC) została dodana niedługo później. Obsługa czwartej architektury sprzętowej, procesora Motorola PowerPC, została dodana w Windows 3.51. Idąc za zmieniającym się zapotrzebowaniem rynku, obsługę procesorów MIPS oraz PowerPC zarzucono przed rozpoczęciem prac nad systemem Windows 2000. Później firma Compaq cofnęła wsparcie dla architektury Alpha AXP, w wyniku czego system Windows 2000 obsługuje jedynie platformę x86.

Kolejną obslugiwaną przez przyszle wersje Windows 2000 architekturą będzie nowa rodzina procesorów Itanium firmy Intel, pierwsza implementacja architektury 64-bitowej, opracowywana wspólnie przez Intela i Hewlett-Packard pod nazwą IA-64 (od Intel Architecture 64). 64-bitowa wersja Windows zapewni o wiele większą przestrzeń adresową zarówno dla procesów użytkownika, jak i dla systemu. Mimo że jest to znaczące usprawnienie, mające pozytywny wplyw na skalowalność systemu, jak dotąd dostosowywanie Windows 2000 do platformy 64-bitowe nie niosło ze sobą konieczności większych zmian w architekturze jądra systemu (innych niż oczywiście zmiany w systemie zarządzania pamięcią). Informacje dotyczące przygotowania obecnie tworzonych programów na nadejście 64-bitowego systemu Windows zawarte są w części dokumentacji pakietu Platform SDK, pod tytułem „Win64 Programming Preview" (dokumentacja ta jest również dostępna pod adresem *msdn.microsoft.com*). Ogólne informacje na temat 64-bitowego systemu Windows można uzyskać po wpisaniu słowa „64-bit" w wyszukiwarce na *www.microsoft. com/windows*.

Windows 2000 zapewnia przenośność pomiędzy architekturami sprzętowymi na dwa główne sposoby:

- Windows 2000 ma strukturę warstwową, w której niskopoziomowe fragmenty systemu, zależne od architektury procesora lub platformy sprzętowej, są odizolowane w postaci oddzielnych modułów, dzięki czemu wyższe warstwy systemu mogą być niezależne od różnic między platformami sprzętowymi. Dwoma kluczowymi komponentami, zapewniającymi przenośność systemu operacyjnego, są jądro (zawarte w pliku *Ntoskrnl.exe*) oraz warstwa uniezależnienia od sprzętu (ang. *Hardware Abstraction Layer* — HAL, zawarta w pliku *Hal.dll*). (Obydwa te komponenty zostaną opisane bardziej szczegółowo w dalszej części

tego rozdziału). Funkcje zależne od architektury (takie jak przełączanie kontekstu wątków oraz obsługa pułapek) są zaimplementowane w jądrze. Funkcje, które mogą różnić się w obrębie tej samej architektury (na przykład różne płyty główne), są zaimplementowane w warstwie HAL.

- Większość systemu Windows 2000 została napisana w języku C, niektóre fragmenty w C++. Asembler stosowano jedynie w tych częściach systemu, które muszą bezpośrednio komunikować się ze sprzętem (na przykład procedury obsługi pułapek i przerwań) lub są krytyczne z punktu widzenia wydajności (jak przełączanie kontekstów). Asembler był stosowany nie tylko w jądrze i warstwie HAL, ale także w kilku innych komponentach systemu (jak na przykład procedury wykorzystujące instrukcje przetwarzania potokowego czy implementujące jeden moduł w podsystemie wywołań procedur lokalnych), w części podsystemu Win32 działającej w trybie jądra, a nawet w niektórych bibliotekach trybu użytkownika, jak kod startu procesu w *Ntdll.dll* (bibliotece systemowej, która zostanie omówiona później).

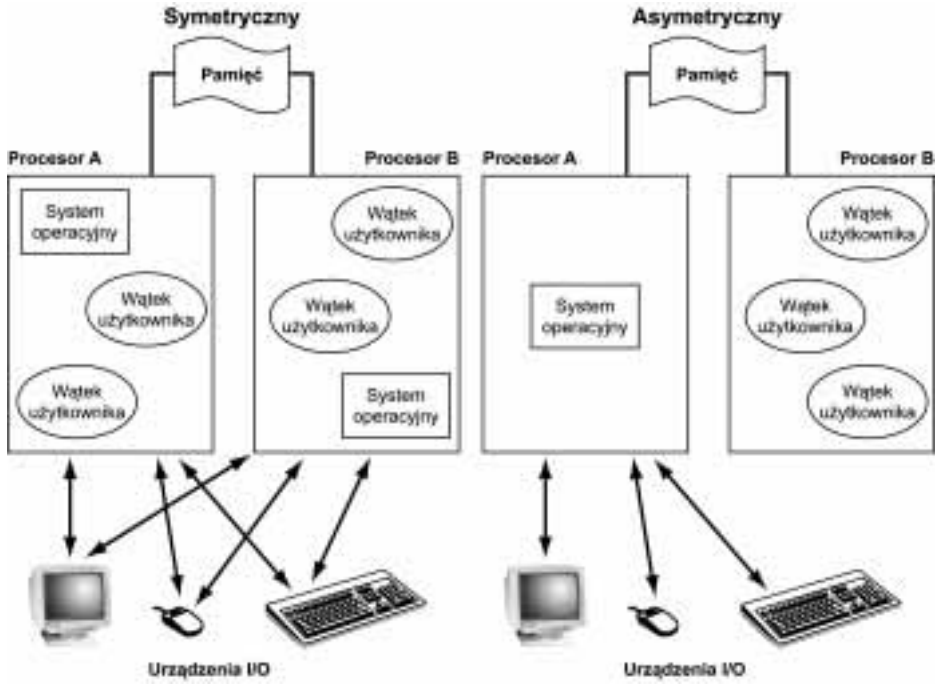
Symetryczne wieloprzetwarzanie

Wielozadaniowość jest techniką, dzięki której system operacyjny umożliwia wykonywanie wielu wątków na jednym procesorze. Jednak, jeśli komputer posiada więcej niż jeden procesor, może wykonywać dwa wątki jednocześnie. Tak więc, podczas gdy wielozadaniowy system operacyjny wygląda tak, jakby wykonywał wiele wątków w tym samym czasie, system z wieloprzetwarzaniem rzeczywiście to robi, wykonując po jednym wątku na każdym z procesorów.

Jak już wspomnieliśmy na początku rozdziału, jednym z kluczowych założeń Windows NT była dobra obsługa systemów wieloprocesorowych. Windows 2000 jest systemem operacyjnym zapewniającym *symetryczne wieloprzetwarzanie* (ang. *simultaneous multiprocessing — SMP*). Nie istnieje procesor nadrzędny — zarówno system operacyjny, jak i wątki użytkownika mogą być wykonywane na dowolnym procesorze. Wszystkie procesory współdzielą także tę samą przestrzeń adresową. Model ten jest przeciwieństwem *asymetrycznego wieloprzetwarzania* (ang. *asymmetric multiprocessing — ASMP*), w którym system operacyjny wybiera jeden procesor, na którym wykonuje swój kod, podczas gdy pozostałe procesory wykonują jedynie kod użytkownika. Różnice między dwoma modelami wieloprzetwarzania ilustruje rysunek 2.1.

Mimo że Windows NT zaprojektowano z myślą o obsłudze do 32 procesorów, żadna z zasadniczych cech systemu nie ogranicza liczby procesorów do 32 — liczba ta jest oczywistą i wygodną konsekwencją długości standardowego słowa 32-bitowego, stosowanego w roli maski bitowej.

Rzeczywista liczba obsługiwanych procesorów zależy od wersji Windows 2000. (Różne wersje Windows 2000 są opisane w następnym podrozdziale). Liczba ta jest przechowywana w rejestrze, w kluczu *HKLM\SYSTEM\CurrentControlSet\Control\Session\Manager\LicensedProcessors*. Warto pamiętać, że modyfikacja tej wartości jest pogwałceniem licencji systemu i prawdopodobnie spowoduje załamanie systemu przy ponownym uruchomieniu, ponieważ modyfikacja rejestru celem umożliwienia obsługi większej liczby procesorów wymaga nie tylko zmiany tej wartości.



Rysunek 2.1. *Wieloprzetwarzanie symetryczne kontra asymetryczne*

Skalowalność

Jednym z kluczowych zagadnień związanych z systemami wieloprocesorowymi jest skalowalność. Aby prawidłowo działać na systemie SMP, kod systemu operacyjnego musi odpowiadać ścisłym wytycznym i zasadom. Przydział zasobów oraz inne kwestie wydajnościowe są w systemach wieloprocesorowych bardziej skomplikowane niż w jednoprocesorowych i muszą być wzięte pod uwagę przy projektowaniu systemu. Windows 2000 posiada kilka cech zapewniających sukces na polu wieloprocesorowych systemów operacyjnych:

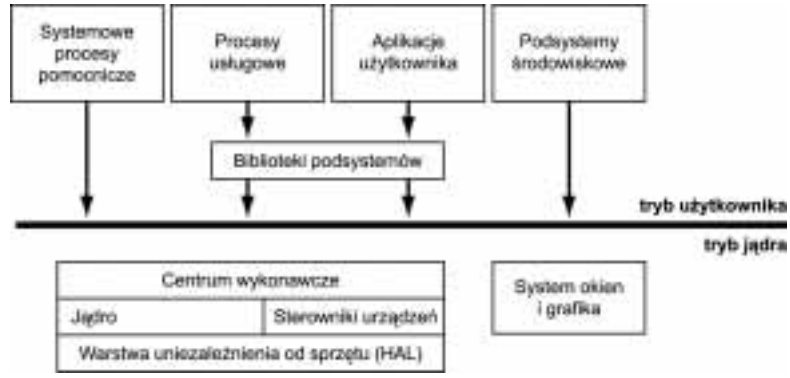
- Zdolność wykonywania kodu systemu operacyjnego na dowolnym dostępnym procesorze lub na wielu procesorach jednocześnie.
- Wiele wątków wykonywanych w ramach jednego procesu — mogą one być równolegle przydzielone do różnych procesorów.
- Drobnoziarnista synchronizacja procesów zarówno jądra, jak i sterowników urządzeń oraz serwerów, która pozwala większej liczbie komponentów działać równolegle na wielu procesorach.

Dodatkowo, Windows 2000 zapewnia mechanizmy (takie jak uzupełniające porty I/O — omówione w rozdziale 9.) umożliwiające efektywną implementację wielowątkowych procesów serwerowych, które mogą być skalowane na systemach wieloprocesorowych.

Synchronizacja wieloprocesorowa zostanie omówiona w rozdziale 3. Szczegóły szeregowania wątków dla wielu procesorów są opisane w rozdziale 6.

Zarys architektury

Po tym krótkim omówieniu założeń projektowych Windows 2000 pora skupić uwagę na kluczowych komponentach składających się na jego architekturę. Uproszczony schemat tej architektury ilustruje rysunek 2.2. Warto pamiętać, że ten diagram jest elementarny — nie zawiera wszystkich informacji. Różne komponenty systemu Windows 2000 zostaną szczegółowo omówione w dalszej części tego rozdziału.



Rysunek 2.2. *Uproszczona architektura Windows 2000*

Na rysunku 2.2 natychmiast rzuca się w oczy linia oddzielająca fragmenty systemu Windows 2000 działające w trybach użytkownika i jądra. Prostokąty ponad linią reprezentują procesy trybu użytkownika, zaś komponenty pod linią są usługami systemu operacyjnego działającymi w trybie jądra. Jak wspomniano w rozdziale 1., wątki trybu użytkownika wykonywane są w chronionej przestrzeni adresowej (choć podczas wykonywania w trybie jądra mają dostęp do przestrzeni systemowej). Tak więc, systemowe procesy pomocnicze, procesy usługowe, aplikacje użytkownika oraz podsystemy środowiskowe posiadają swoje własne, prywatne przestrzenie adresowe.

Cztery podstawowe typy procesów użytkownika są opisane poniżej:

- Stałe *systemowe procesy pomocnicze* (ang. *system support processes*), takie jak proces logowania oraz menedżer sesji, które nie są usługami Windows 2000 (tzn. nie są uruchamiane przez menedżera kontroli usług).
- *Procesy usługowe* zapewniające usługi Win32, takie jak mechanizm szeregowania zadań (*Task Manager*) czy *Spooler*. Wiele aplikacji serwerowych dla Windows 2000, jak na przykład Microsoft SQL Server czy Microsoft Exchange Server, również zawiera komponenty działające jako usługi.
- *Aplikacje użytkownika*, które mogą należeć do pięciu rodzajów: Win32, Windows 3.1, MS-DOS, POSIX lub OS/2 1.2.
- *Podsystemy środowiskowe*, umożliwiające aplikacjom użytkownika dostęp do wewnętrznych usług systemu operacyjnego poprzez zestaw funkcji, tworzących *środowisko* systemu operacyjnego. Windows 2000 zawiera trzy podsystemy środowiskowe: Win32, POSIX oraz OS/2.

Na rysunku 2.2 zwraca uwagę komponent o nazwie „Biblioteki podsystemów" poniżej prostokątów „Procesy usługowe" i „Aplikacje użytkownika". W Windows 2000 aplikacje użytkownika nie wywołują wbudowanych usług systemu operacyjnego bezpośrednio. Zamiast tego korzystają z jednej lub więcej *podsystemowych bibliotek dynamicznych* (ang. *subsystem dynamic-link libraries — DLL*). Zadaniem biblioteki podsystemowej jest przetłumaczenie udokumentowanej funkcji na odpowiednie wywołanie wewnętrznej (i nieudokumentowanej) usługi systemowej Windows 2000. To tłumaczenie może, ale nie musi, być związane z wysłaniem komunikatu do procesu podsystemu środowiskowego, wykorzystywanego przez aplikację.

Do komponentów Windows 2000 działających w trybie jądra należą:

- *Centrum wykonawcze* (ang. *executive*) Windows 2000 zawiera podstawowe usługi systemu operacyjnego, jak zarządzanie pamięcią, zarządzanie procesami i wątkami, bezpieczeństwo, I/O oraz komunikacja międzyprocesowa.
- *Jądro* (ang. *kernel*) Windows 2000 zawiera niskopoziomowe funkcje systemu operacyjnego, takie jak szeregowanie wątków, obsługa przerwań i wyjątków oraz synchronizacja wieloprocesorowa. Zapewnia również zestaw procedur i podstawowych obiektów wykorzystywanych przez centrum wykonawcze do implementacji konstrukcji wyższego poziomu.
- Do *sterowników urządzeń* (ang. *device drivers*) należą zarówno sterowniki urządzeń sprzętowych, tłumaczące wywołania funkcji I/O na konkretne operacje I/O zależne od fizycznego urządzenia, jak i systemy plików oraz sterowniki sieciowe.
- *Warstwa uniezależnienia od sprzętu* (ang. *hardware abstraction layer — HAL*) jest warstwą kodu uniezależniającą jądro, sterowniki urządzeń oraz pozostałą część centrum wykonawczego Windows 2000 od zróżnicowania warstwy sprzętowej (jak na przykład różnice między płytami głównymi).
- *System okien i grafiki* implementuje funkcje graficznego interfejsu użytkownika (lepiej znane jako funkcje podsystemów Win32 USER i GDI), związane z zarządzaniem oknami, kontrolkami interfejsu użytkownika oraz rysowaniem.

Tabela 2.1 zawiera nazwy plików podstawowych komponentów systemu operacyjnego Windows 2000. (Nazwy te warto znać, ponieważ w książce znajdują się odwołania do plików systemowych). Każdy z tych komponentów jest omówiony bardziej szczegółowo w dalszej części tego rozdziału oraz w rozdziałach następnych.

Jednak przed zagłębieniem się w szczegóły tych komponentów systemowych warto zbadać różnice pomiędzy systemem w wersji Windows 2000 Professional, a różnymi wersjami Windows 2000 Server.

Pakiety handlowe Windows 2000

Istnieją cztery edycje systemu Windows 2000: Windows 2000 Professional, Windows 2000 Server, Windows 2000 Advanced Server oraz Windows 2000 Datacenter Server. Wersje te różnią się:

Tabela 2.1. *Podstawowe pliki systemowe Windows 2000*

Nazwa pliku	Komponenty
Ntoskrnl.exe	Centrum wykonawcze i jądro
Ntkrnlpa.exe	Centrum wykonawcze i jądro obsługujące rozszerzenie adresu fizycznego (PAE), pozwalające na adresowanie do 64 GB pamięci fizycznej
Hal.dll	Warstwa uniezależnienia od sprzętu
Win32k.sys	Część podsystemu Win32 działająca w trybie jądra
Ntdll.dll	Wewnętrzne funkcje wspomagające oraz systemowe łączniki do funkcji wykonawczych
Kernel32.dll, *Advapi32.dll*, *User32.dll*, *Gdi32.dll*	Podstawowe biblioteki podsystemu Win32

- Liczbą obsługiwanych procesorów.
- Wielkością obsługiwanej pamięci fizycznej.
- Liczbą obsługiwanych jednocześnie połączeń sieciowych.
- Warstwami usługowymi, dołączanymi do edycji Server, ale nie do edycji Professional.

Różnice te są zestawione w tabeli 2.2.

Komponentami, które *nie* różnią się pomiędzy różnymi wersjami Windows 2000, są podstawowe pliki systemowe: kod jądra, *Ntoskrnl.exe* (oraz wersja PAE, *Ntkrnlpa.exe*); biblioteki warstwy HAL, sterowniki urządzeń oraz podstawowe narzędzia i biblioteki systemowe. Wszystkie te pliki są takie same dla wszystkich edycji Windows 2000. Na przykład nie istnieją specjalne, serwerowe wersje warstwy HAL.

Jednakże niektóre z tych komponentów działają różnie, w zależności od wersji systemu, z którą współpracują. Systemy Windows 2000 Server są optymalizowane ze względu na maksymalną przepustowość, konieczną w roli wysokowydajnych serwerów aplikacyjnych, podczas gdy Windows 2000 Professional, mimo iż posiada możliwości serwerowe, jest optymalizowany ze względu na czas reakcji przy pracy z użytkownikiem. Na przykład na bazie typu produktu, podczas rozruchu systemu podejmowane są różne decyzje dotyczące przydziału zasobów, takich jak liczba i rozmiar stert systemowych (lub pul), liczba wewnętrznych, systemowych wątków roboczych oraz rozmiar systemowej podręcznej pamięci danych. Również decyzje podejmowane podczas pracy systemu, jak sposób obsługi pochodzących od systemu i procesów żądań dotyczących pamięci, różnią się pomiędzy edycjami Windows 2000 Server i Windows 2000 Professional. Nawet pewne szczegóły szeregowania wątków są różne między tymi dwiema rodzinami systemów. Jeśli istnieją znaczące różnice w działaniu pomiędzy dwiema edycjami, są one wyszczególnione w odpowiednich rozdziałach tej książki. O ile nie zaznaczymy inaczej, wszystkie informacje zawarte w tej książce dotyczą zarówno edycji Windows 2000 Server, jak i Windows 2000 Professional.

Skoro kod jądra jest identyczny we wszystkich edycjach Windows 2000, to skąd system „wie", która wersja została uruchomiona? Poprzez wartości zapisane w rejestrze pod nazwami `ProductType` i `ProductSuite`, w kluczu *HKLM\SYSTEM\CurrentControlSet*

Tabela 2.2. *Różnice między wersjami Windows 2000 Professional i Server*

Edycja	Liczba obslugiwanych procesorów	Obslugiwana pamięć fizyczna	Liczba jednoczesnych, klienckich połączeń sieciowych	Dodatkowe warstwy uslugowe
Windows 2000 Professional	2	4 GB	10*	
Windows 2000 Server	4	4 GB	Nieograniczona	Możliwość prowadzenia kontrolera domeny, usługa katalogowa, programowy system RAID, serwer protokołu dynamicznej konfiguracji serwerów (ang. *Dynamic Host Configuration Protocol — DHCP*), serwer rozproszonego systemu plików (ang. *Distributed File System — DFS*), usługi certyfikacyjne, usługi zdalnej instalacji oraz terminali
Windows 2000 Advanced Server	8	8 GB	Nieograniczona	*dwuwęzłowy klaster*
Windows 2000 Datacenter Server	32	64 GB**	Nieograniczona	*dwuwęzłowy klaster*, narzędzie menedżera kontroli procesów***

* Ustęp w Umowie Licencyjnej użytkownika Windows 2000 Professional (zawartej w pliku *\Winnt\System32\Eula.txt*) mówi: „Licencjobiorca może zezwolić na połączenie ze Stacją Roboczą co najwyżej dziesięciu (10) komputerów lub innych urządzeń elektronicznych (każde z nich będzie nazywane „Urządzeniem") w celu uzyskania dostępu i korzystania z usług Produktu, wyłącznie takich, jak usługi plikowe i usługi drukowania, internetowe usługi informacyjne oraz usługi dostępu zdalnego (łącznie ze wspólnym korzystaniem z połączeń i usługami telefonicznymi)". Ograniczenie to dotyczy współdzielenia plików i wydruków oraz dostępu zdalnego, ale nie do internetowych usług informacyjnych (ang. *Internet Information Services*).

** Ograniczenie teoretyczne — rzeczywisty limit może być mniejszy, ze względu na dostępność sprzętu.

*** Więcej informacji na temat narzędzia menedżera kontroli procesów znajduje się w rozdziale 6.

Control\ProductOptions. `ProductType` określa, czy system jest wersją Windows 2000 Professional, czy Windows 2000 Server (dowolnej edycji). Dopuszczalne wartości tej zmiennej są podane w tabeli 2.3. Rezultat jest zapisany w globalnej zmiennej systemowej `MmProductType`, dostępnej dla sterowników urządzeń dzięki funkcji trybu jądra, o nazwie `MmIsThisAnNtAsSystem`, udokumentowanej w pakiecie Windows 2000 DDK.

Tabela 2.3. *Wartości zmiennej rejestrowej ProductType*

Edycja Windows 2000	Wartość ProductType
Windows 2000 Professional	`WinNT`
Windows 2000 Server (kontroler domeny)	`LanmanNT`
Windows 2000 Server (tylko serwer)	`ServerNT`

Inna zmienna rejestru, ProductSuite, pozwala rozróżnić edycje Windows 2000 Server, Advanced Server oraz Datacenter Server, jak również określa, czy zainstalowano usługi terminalowe (w przypadku systemów Server). W systemach Windows 2000 Professional ta zmienna jest pusta.

Programy użytkownika, aby określić, na której edycji Windows 2000 są uruchomione, mogą skorzystać z funkcji Win32 o nazwie VerifyVersionInfo, opisanej w dokumentacji pakietu Platform SDK. Sterowniki urządzeń mogą skorzystać z funkcji trybu jądra, o nazwie RtlGetVersion, udokumentowanej w Windows 2000 DDK.

Wersja testowa

Istnieje specjalna, wspomagająca wykrywanie błędów, edycja Windows 2000 Professional nosząca nazwę *wersji testowej* (ang. *checked build*). Wersja ta jest dostępna jedynie z prenumeratą biblioteki MSDN Professional (lub Universal). Jej celem jest wsparcie projektantów sterowników urządzeń — wersja testowa zapewnia bardziej ścisłe wychwytywanie błędów w funkcjach trybu jądra, wywoływanych przez sterowniki urządzeń i inny kod systemowy. Na przykład, jeśli sterownik (lub inny fragment kodu w trybie jądra) wykonuje błędne wywołanie funkcji, która sprawdza poprawność parametrów (jak na przykład żądanie spinlock na niewłaściwym poziomie przerwań), system przerwie wykonywanie w momencie wykrycia błędu, nie dopuszczając do zakłócenia integralności struktur danych i ewentualnego, późniejszego załamania systemu.

Wersja testowa jest efektem rekompilacji kodu źródłowego Windows 2000 z parametrem kompilacji DEBUG ustawionym na TRUE. Wiele dodatkowego kodu w plikach powstało w rezultacie zastosowania makra ASSERT, zdefiniowanego w pliku nagłówkowym *Ntddk.h* i udokumentowanego w materiałach dostarczanych z DDK. Makro to sprawdza warunek (taki jak poprawność struktury danych lub parametru), a jeśli sprawdzane wyrażenie ma wartość logiczną FALSE, makro wywołuje funkcję trybu jądra RtlAssert, która korzysta z funkcji DbgPrint celem przekazania tekstu komunikatu o błędzie do programu uruchomieniowego jądra (jeśli takowy jest obecny), po czym „pyta" użytkownika co robić (przerwać, zignorować, zakończyć proces czy zakończyć wątek). Jeśli system nie został uruchomiony z programem uruchomieniowym jądra (przez zastosowanie opcji /DEBUG w pliku *Boot.ini*) i żaden taki program aktualnie nie działa, porażka testu ASSERT powoduje załamanie systemu.

Mimo że Microsoft nie rozpowszechnia wersji testowej systemów Windows 2000 Server, Advanced Server ani Datacenter Server, można ręcznie zastąpić kod jądra Windows 2000 Server wersją testową i ponownie uruchomić system z testowym jądrem. (Można tak zrobić z pozostałymi plikami systemowymi, ale większość projektantów korzystających z testowej edycji potrzebuje jedynie rozszerzonej wersji jądra — a nie wszystkich sterowników urządzeń, narzędzi i bibliotek).

Pliki systemowe związane z maszynami wieloprocesorowymi

Sześć spośród plików systemowych[1] jest innych w systemie wieloprocesorowym niż w jednoprocesorowym. (zob. tabela 2.4.) Podczas instalacji odpowiedni plik jest kopiowany

[1] W katalogu *\I386\UNIPROC* na płycie dystrybucyjnej Windows 2000 znajduje się plik o nazwie *Winsrv.dll* — mimo że ten plik znajduje się w katalogu o nazwie *UNIPROC*, co sugeruje, że istnieje wersja jednoprocesorowa, w rzeczywistości istnieje tylko jedna wersja tego pliku dla systemów jedno- i wieloprocesorowych.

do lokalnego katalogu *\Winnt\System32*. Aby sprawdzić, które pliki zostały skopiowane, można obejrzeć zawartość pliku *\Winnt\Repair\Setup.log*, zawierającego nazwy wszystkich plików skopiowanych na lokalny dysk systemowy, wraz ze ścieżką źródłową tych plików.

Tabela 2.4. *Pliki systemu wieloprocesorowego kontra pliki systemu jednoprocesorowego*

Nazwa pliku na dysku systemowym	Nazwa wersji jednoprocesorowej na plycie CD	Nazwa wersji wieloprocesorowej na plycie CD
Ntoskrnl.exe	*\I386\Ntoskrnl.exe*	*\I386\Ntkrnlmp.exe*
Ntkrnlpa.exe	*Ntkrnlpa.exe* w *\I386\Driver.cab*	*Ntkrpamp.exe* w *\I386\Driver.cab*
Hal.dll	Zależnie od typu systemu (lista wersji HAL znajduje się w tabeli 2.5.)	Zależnie od typu systemu (lista wersji HAL znajduje się w tabeli 2.5.)
Win32k.sys	*\I386\UNIPROC\Win32k.sys*	*\I386\Win32k.sys*
Ntdll.dll	*\I386\UNIPROC\Ntdll.dll*	*\I386\Ntdll.dll*
Kernel32.dll	*\I386\UNIPROC\Kernel32.dll*	*\I386\Kernel32.dll*

Powodem istnienia jednoprocesorowych wersji tych kluczowych plików systemowych jest wydajność — synchronizacja wieloprocesorowa jest z natury bardziej złożona i czasochłonna niż korzystanie z jednego procesora, tak więc specjalne, jednoprocesorowe wersje kluczowych plików pozwalają uniknąć tego nakładu w systemach z jednym procesorem (które składają się na większość komputerów pracujących pod kontrolą Windows 2000).

Interesujące jest, że podczas gdy jedno- i wieloprocesorowe wersje pliku *Ntoskrnl* powstały poprzez kompilację kodu źródłowego z odpowiednimi opcjami, to jednoprocesorowe wersje plików *Ntdll.dll* i *Kernel32.dll* powstały poprzez zastąpienie instrukcji `LOCK` oraz `UNLOCK`, używanych w architekturze x86 do synchronizacji wątków, instrukcjami `NOP` (które nic nie robią).

ĆWICZENIE: Obserwacja plików związanych z systemem wieloprocesorowym

Pliki, których inne wersje są wymagane dla systemu wieloprocesorowego, można obejrzeć, przeglądając szczegóły sterowników w *Menedżerze urządzeń*:

1. Otwórz *Właściwości* systemu (poprzez wybranie ikony *System* w *Panelu Sterowania* lub przez naciśnięcie prawego klawisza myszy na ikonie *Mój Komputer* na pulpicie i wybranie *Właściwości*).
2. Wybierz zakładkę *Sprzęt*.
3. Wybierz *Menedżer urządzeń*.
4. Rozwiń obiekt *Komputer*.
5. Podwójnie naciśnij uchwyt poniżej pola *Komputer*.
6. Wybierz zakładkę *Sterownik*.
7. Naciśnij *Szczegóły sterownika*

Pozostałe pliki systemowe składające się na Windows 2000 (włącznie z narzędziami, bibliotekami i sterownikami urządzeń) występują w tej samej wersji w systemach jedno- i wieloprocesorowych (co oznacza, że prawidłowo współpracują z komputerami wieloprocesorowymi). To podejście warto stosować podczas pracy nad każdym oprogramowaniem, zarówno w Win32, jak i przy okazji projektowania sterowników — musisz brać pod uwagę kwestie związane z wieloprocesorowością i testować gotowe programy na systemach jedno- i wieloprocesorowych.

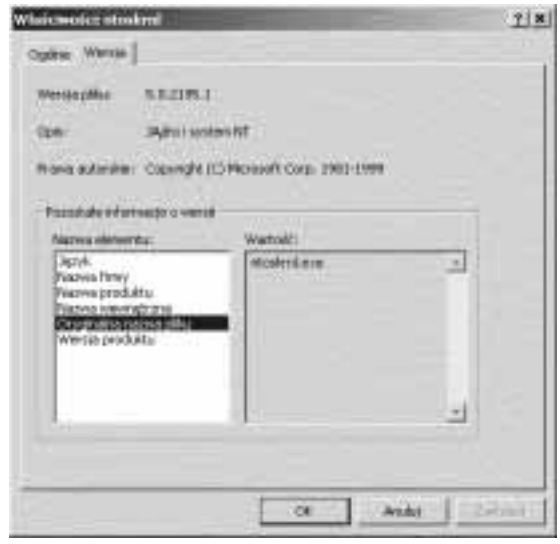
Na płycie zawierającej wersję testową Windows 2000 porównanie plików *Ntoskrnl.exe* i *Ntkrnlmp.exe* lub *Ntkrnlpa.exe* i *Ntkrpamp.exe* wykaże, że są one identyczne — są to wieloprocesorowe wersje tych samych plików. Innymi słowy, nie istnieją jednoprocesorowe wersje testowe jąder dostarczanych z wersją „checked build".

ĆWICZENIE: Sprawdzenie wersji uruchomionego pliku Ntoskrnl

W przypadku Windows NT można sprawdzić wersję uruchomionego pliku *Ntoskrnl* poprzez naciśnięcie zakładki *Wersja* w programie Diagnostyka Windows NT (*Winmsd.exe*). W Windows 2000 nie ma narzędzia pozwalającego uzyskać tę informację. Jednakże podczas każdego rozruchu systemu dokonywany jest wpis do *Dziennika zdarzeń*, zapisujący typ jądra (jedno- lub wieloprocesorowe oraz zwykłe lub testowe). Jest to zilustrowane na rysunku poniżej. (Z menu *Start* wybierz *Programy/Narzędzia administracyjne/Podgląd zdarzeń*, wybierz *Dziennik systemu*, a następnie podwójnie naciśnij wpis *Dziennika* o identyfikatorze 6009, oznaczającym, że wpisu dokonano przy rozruchu systemu).

Ten zapis w *Dzienniku* nie określa, czy uruchomiono wersję jądra zapewniającą PAE — możliwość obsługi pamięci fizycznej większej niż 4 GB (*Ntkrnlpa.exe*). Jednak ta informacja znajduje się rejestrze, w zmiennej o nazwie *HKLM\SYSTEM\CurrentControlSet\Control\SystemStartOptions*. W przypadku uruchomienia jądra PAE, zmienna *HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\PhysicalAddressExtension* otrzymuje wartość 1.

Można również określić, czy uruchomiono wieloprocesorową wersję *Ntoskrnl* (lub *Ntkrnlpa*). W tym celu trzeba sprawdzić właściwości plików: uruchom Windows Explorera, prawym klawiszem myszy naciśnij na pliku *Ntoskrnl.exe* w katalogu *\Winnt\System32*, po czym wybierz *Właściwości*. Następnie naciśnij zakładkę *Wersja* i wybierz *Oryginalna nazwa pliku* — w przypadku wersji jednoprocesorowej ukaże się okno dialogowe jak na poniższym rysunku.

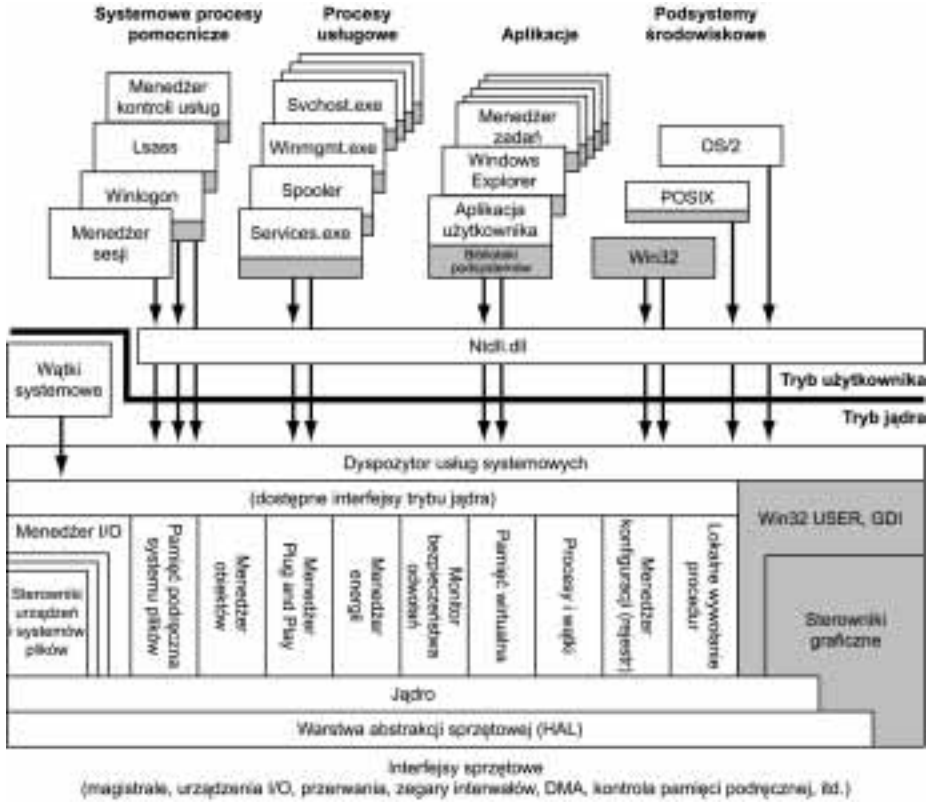


Wreszcie, jak wspomniano wcześniej, można sprawdzić, które wersje jądra i warstwy HAL zostały wybrane na etapie instalacji. Sprawdzisz to poprzez obejrzenie zawartości pliku *\Winnt\Repair\Setup.log*.

Kluczowe komponenty systemu

Po omówieniu wysokopoziomowej architektury Windows 2000 pora zagłębić się w wewnętrzną strukturę i zadania realizowane przez poszczególne, kluczowe komponenty systemu. Rysunek 2.3 jest bardziej szczegółową i kompletną wersją diagramu komponentów i architektury Windows 2000 zamieszczonego wcześniej (na rysunku 2.2).

Poniższe podrozdziały prezentują większe elementy tego diagramu. W rozdziale 3. wyjaśniamy główne mechanizmy kontrolne systemu (jak menedżer obiektów, przerwania, itd.). W rozdziale 4. przedstawiamy proces rozruchu i zamykania Windows 2000, natomiast rozdział 5. zawiera szczegóły mechanizmów zarządzających, takich jak rejestr, procesy usługowe oraz instrumentacje zarządzania Windows (ang. *Windows Management Instrumentation — WMI*). Kolejne rozdziały pokazują jeszcze bardziej szczegółowo wewnętrzną strukturę i działanie kluczowych mechanizmów, takich jak procesy i wątki, zarządzanie pamięcią, bezpieczeństwo, zarządzanie urządzeniami I/O, zarządzanie pamięcią masową, zarządzanie pamięcią podręczną, system plików Windows 2000 (NTFS) oraz usługi sieciowe.



Rysunek 2.3. *Architektura Windows 2000*

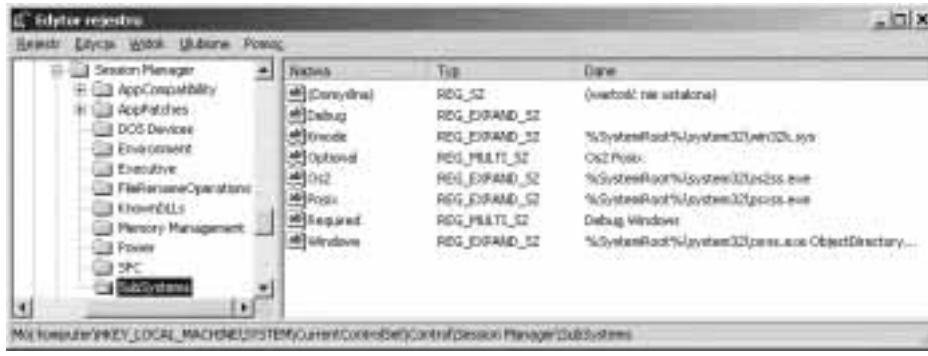
Podsystemy środowiskowe i biblioteki podsystemów

Windows 2000 posiada trzy podsystemy środowiskowe: OS/2, POSIX oraz Win32 (ilustruje to rysunek 2.3). Jak się zaraz okaże, spośród tych trzech Win32 jest podsystemem wyjątkowym, ponieważ Windows 2000 nie może bez niego działać. (Obsługuje on klawiaturę, mysz oraz ekran, i musi być obecny nawet w systemach serwerowych pozbawionych możliwości interaktywnej pracy z użytkownikiem). Dwa pozostałe podsystemy są skonfigurowane tak, że uruchamiają się na żądanie, podczas gdy Win32 musi działać zawsze.

Informacja o rozruchu podsystemów znajduje się w kluczu rejestru o nazwie *HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\SubSystems*. Rysunek 2.4 ilustruje wartości w tym kluczu.

Pole `Required` zawiera podsystemy ładowane podczas rozruchu systemu. Zmienna ta zawiera dwa ciągi znaków: `Windows` oraz `Debug`. Pole `Windows` zawiera specyfikację pliku podsystemu Win32, *Csrss.exe* (skrót pochodzi od *Client/Server Run-Time Subsystem*)[2].

[2] **Nota historyczna: powód, dla którego proces podsystemu Win32 nosi nazwę** *Csrss.exe*, **pochodzi z czasów początkowego projektu Windows NT, gdzie wszystkie podsystemy miały działać jako wątki pojedynczego, ogólnego procesu podsystemów środowiskowych. Gdy podsystemy POSIX i OS/2 umieszczono we własnych procesach, nazwa procesu podsystemu Win32 pozostała bez zmian.**



Rysunek 2.4. *Edytor rejestru pokazujący informację o rozruchu Windows 2000*

Pole `Debug` jest puste (służy do wewnętrznego testowania) i nie ma żadnych efektów. Wartość `Optional` oznacza, że podsystemy OS/2 i POSIX będą uruchamiane na żądanie. Zmienna rejestrowa `Kmode` zawiera nazwę pliku części podsystemu Win32 trybu jądra, *Win32k.sys* (omawianego w dalszej części rozdziału).

Zadaniem podsystemu środowiskowego jest zapewnienie dostępu do pewnego zbioru podstawowych usług systemowych Windows 2000 dla aplikacji. Każdy podsystem zapewnia dostęp do innego zestawu wewnętrznych usług Windows 2000. Oznacza to, że pewnych rzeczy, których można dokonać w aplikacji zbudowanej z wykorzystaniem jednego z podsystemów, nie można zrealizować w aplikacji powstałej dla innego podsystemu. Na przykład aplikacje Win32 nie mogą korzystać z funkcji `fork`, należącej do podsystemu POSIX.

Każdy plik wykonywalny (*.exe*) jest przypisany do jednego i tylko jednego podsystemu. W momencie uruchomienia pliku kod tworzący proces analizuje typ podsystemu zapisany w nagłówku pliku, dzięki czemu może powiadomić odpowiedni podsystem. Ten znacznik typu jest określany poprzez opcję `/SUBSYSTEM` polecenia `link` w pakiecie Microsoft Visual C++ i może być sprawdzony za pomocą narzędzia *Exetype* z Windows 2000 Resource Kit.

Wywołania funkcji nie mogą mieszać podsystemów. Innymi słowy, aplikacja POSIX może wywoływać jedynie usługi udostępniane przez podsystem POSIX, a aplikacja Win32 może korzystać jedynie z usług udostępnianych przez podsystem Win32. Jak się okaże, ograniczenie to jest powodem, dla którego podsystem POSIX, implementujący bardzo ograniczony zestaw funkcji (tylko POSIX 1003.1), nie jest przydatnym środowiskiem do przenoszenia aplikacji z systemów Uniks.

Jak już wspomniano, aplikacje użytkownika nie wywołują usług systemowych Windows 2000 bezpośrednio. Zamiast tego korzystają z jednej lub więcej bibliotek podsystemów. Biblioteki te oferują udokumentowany interfejs, dostępny dla programów korzystających z tych bibliotek. Na przykład biblioteki podsystemu Win32 (jak *Kernel32.dll*, *Advapi32.dll*, *User32.dll* oraz *Gdi32.dll*) implementują funkcje Win32 API. Biblioteka podsystemu POSIX implementuje interfejs programowy POSIX 1003.1.

Gdy aplikacja wywołuje funkcję z biblioteki podsystemu, możliwe są trzy rzeczy:

- Funkcja jest w całości zaimplementowana w trybie użytkownika w bibliotece podsystemowej. Innymi słowy, nie zostaje wysłany żaden komunikat do procesu podsystemu ani nie jest wywoływana żadna wewnętrzna usługa systemowa Windows 2000. Funkcja zostaje wykonana w trybie użytkownika, a rezultat zostaje zwrócony do procesu wywołującego. Do takich funkcji należy `GetCurrentProcess` (która zawsze zwraca –1, wartość oznaczającą aktualny proces dla wszystkich funkcji związanych z procesami) oraz `GetCurrentProcessId` (identyfikator procesu nie zmienia się podczas wykonywania procesu, tak więc pobierany jest spośród danych lokalnych, bez potrzeby wywoływania jądra).
- Funkcja wymaga jednego lub więcej wywołań centrum wykonawczego Windows 2000. Na przykład funkcje Win32 o nazwach `ReadFile` i `WriteFile` związane są z wywoływaniem wewnętrznych (i nieudokumentowanych) usług systemu I/O Windows 2000, odpowiednio `NtReadFile` i `NtWriteFile`.
- Funkcja wymaga wykonania pewnych operacji przez proces podsystemu środowiskowego. (Procesy podsystemów środowiskowych, działające w trybie użytkownika, są odpowiedzialne za stan aplikacji klienta, działającej pod ich kontrolą). W tym przypadku wysuwane jest za pośrednictwem komunikatu do podsystemu żądanie typu klient-serwer, wymuszające wykonanie jakiejś operacji. Biblioteka podsystemu następnie czeka na odpowiedź, po czym powraca do programu wywołującego.

Niektóre funkcje mogą być kombinacjami drugiego i trzeciego spośród powyższych wariantów. Tak jest w przypadku funkcji Win32 o nazwach `CreateProcess` oraz `CreateThread`.

ĆWICZENIE: Oglądanie typu podsystemu pliku wykonywalnego

Typ podsystemu pliku wykonywalnego można sprawdzić za pomocą narzędzia *Exetype* z Windows 2000 Resource Kit lub narzędzia *Dependency Walker* (*Depends.exe*) z pakietów Windows 2000 Support Tools i Platform SDK. Na przykład poniżej znajdują się typy plików wykonywalnych dwóch różnych programów Win32, *Notepad.exe* (prostego edytora tekstu) oraz *Cmd.exe* (wiersza poleceń Windows 2000):

```
C:\>exetype \winnt\system32\notepad.exe
File "\winnt\system32\notepad.exe" is of the following type:
    Windows NT
    32 bit machine
    Built for the Intel 80386 processor
    Runs under the Windows GUI subsystem

C:\>exetype \winnt\system32\cmd.exe
File "\winnt\system32\cmd.exe" is of the following type:
    Windows NT
    32 bit machine
    Built for the Intel 80386 processor
    Runs under the Windows character-based subsystem
```

W rzeczywistości istnieje tylko jeden podsystem Windows, nie ma oddzielnych wersji: graficznej i tekstowej. Poza tym, Windows 2000 nie może być uruchomiony na procesorze Intel 386 (ani na 486) — komunikat tekstowy programu *Exetype* po prostu dawno nie był uaktualniany.

Mimo że Windows 2000 zaprojektowano z myślą o obsłudze wielu niezależnych podsystemów środowiskowych, z praktycznego punktu widzenia implementacja w każdym z podsystemów kodu odpowiedzialnego za okna i wyświetlanie powodowałaby znaczną redundancję funkcji systemowych, która miałaby negatywny wpływ na rozmiary i wydajność systemu. Ponieważ Win32 miał być głównym podsystemem, projektanci Windows 2000 zdecydowali się na umieszczenie podstawowych funkcji właśnie w nim, podczas gdy inne podsystemy korzystałyby z funkcji wyświetlania podsystemu Win32. Tak więc podsystemy POSIX i OS/2 korzystają z usług podsystemu Win32 związanych z wyświetlaniem. (W rzeczywistości, jeśli sprawdzić typ podsystemu plików z kodem tych bibliotek, okaże się, że są one plikami podsystemu Win32).

Warto przyjrzeć się bliżej każdemu z podsystemów środowiskowych.

Podsystem Win32

Podsystem Win32 składa się z następujących komponentów:

- Proces podsystemu środowiskowego (*Csrss.exe*) zapewnia:
  - Obsługę okien konsoli (tekstowych).
  - Tworzenie oraz niszczenie procesów i wątków.
  - Częściową obsługę procedur 16-bitowej wirtualnej maszyny DOS-a (VDM).
  - Różne funkcje, jak `GetTempFileName`, `DefineDosDevice`, `ExitWindowsEx` oraz kilka funkcji związanych z obsługą języków narodowych.
- Sterownik urządzeń trybu jądra (*Win32k.sys*) zawiera:
  - Menedżera okien, który kontroluje wyświetlanie okien, zarządza ekranem, odbiera dane z klawiatury, myszy i innych urządzeń oraz przekazuje komunikaty użytkownika do aplikacji.
  - Interfejs urządzeń graficznych (ang. *Graphics Device Interface — GDI*), który jest biblioteką funkcji dla graficznych urządzeń wyjściowych. Zawiera funkcje służące do rysowania linii, tekstu i figur oraz do operacji graficznych.
- Biblioteki systemowe (jak *Kernel32.dll*, *Advapi32.dll*, *User32.dll* i *Gdi32.dll*) tłumaczą udokumentowane funkcje Win32 API na odpowiednie wywołania nieudokumentowanych usług systemowych trybu jądra, z plików *Ntoskrnl.exe* i *Win32k.sys*.
- Sterowniki urządzeń graficznych są zależnymi od sprzętu sterownikami kart graficznych, drukarek i miniportów video.

Aplikacje korzystają ze standardowych funkcji z grupy *USER* celem stworzenia na ekranie kontrolek interfejsu użytkownika, jak okna i przyciski. Menedżer okien powiadamia o tych żądaniach moduł GDI, który przekazuje je do sterowników urządzeń graficznych, rysujących na ekranie. Karta graficzna współpracuje ze sterownikiem portu video, co zamyka przepływ danych.

GDI zapewnia zestaw standardowych funkcji grafiki dwuwymiarowej, które pozwalają aplikacjom komunikować się z urządzeniami graficznymi, bez znajomości specyfiki urządzenia. Funkcje GDI są warstwą pośrednią pomiędzy aplikacjami a urządzeniami

graficznymi, takimi jak karty graficzne czy drukarki. GDI interpretuje żądania aplikacji dotyczące grafiki i wysyła żądania do sterowników urządzeń graficznych. Zapewnia również standardowy interfejs, dzięki któremu aplikacje mogą korzystać z różnych urządzeń graficznych. Interfejs ten pozwala na niezależność kodu aplikacji od urządzeń i ich sterowników. GDI dostosowuje swoje komunikaty do możliwości urządzenia, często rozbijając żądania na części składowe. Na przykład niektóre urządzenia potrafią zinterpretować polecenie narysowania elipsy. Inne wymagają, by funkcja GDI przetworzyła to polecenie na ciąg pikseli o odpowiednich współrzędnych. Więcej informacji na temat grafiki i architektury sterowników video znajduje się w rozdziale „Design Guide" książki „Graphics Drivers" z pakietu Windows 2000 DDK.

Przed Windows NT 4 usługi graficzne i menedżer okien wchodziły w skład procesu podsystemu Win32 w trybie użytkownika. W Windows NT 4 większość kodu odpowiedzialnego za okna i grafikę została przeniesiona z procesu podsystemu Win32 do zbioru usług wykonywanych w trybie jądra (w pliku *Win32k.sys*). Głównym powodem tej zmiany było zwiększenie ogólnej wydajności. Oddzielny proces zawierający podsystem graficzny Win32 wymagał wielokrotnego przełączania kontekstów wątków i procesów, co zajmowało znaczną liczbę taktów procesora i pewną ilość pamięci, nawet mimo znacznej optymalizacji oryginalnego projektu.

Na przykład dla każdego wątku po stronie klienta, w procesie podsystemu Win32 istniał osobny, dedykowany wątek serwera, czekający na żądania wątku klienta. Do komunikacji między tymi wątkami wykorzystywano specjalny mechanizm komunikacji międzyprocesowej, nazywany *szybkim LPC*. W przeciwieństwie do zwykłego przełączenia kontekstu wątku, przejścia pomiędzy skojarzonymi wątkami, wykorzystujące szybkie LPC, nie powodują zdarzenia szeregowania wątków jądrze. Pozwala to wątkowi serwera działać w kwancie czasu wątku klienta, zanim zostanie wywłaszczony przez proces szeregujący. Co więcej, wykorzystywano bufory pamięci współdzielonej do szybkiego przekazywania dużych struktur danych, jak obrazy — a klienci mieli możliwość odczytu kluczowych struktur danych serwera, celem zmniejszenia liczby przejść między klientami a serwerem Win32. Operacje GDI były (i wciąż są) przetwarzane wsadowo. *Przetwarzanie wsadowe* (ang. *batching*) oznacza, że ciąg wywołań pochodzących z aplikacji Win32 nie jest „wrzucany" do serwera i nie powoduje rysowania na ekranie, do momentu wypełnienia kolejki modułu GDI. Rozmiar kolejki można określić za pomocą funkcji `GdiSetBatchLimit`, a opróżnić kolejkę w dowolnym momencie można za pomocą `GdiFlush`. Także właściwości tylko do odczytu oraz struktury danych GDI, po pobraniu od procesu podsystemu Win32, były tymczasowo składowane po stronie klienta, celem przyspieszenia dostępu.

Mimo tych optymalizacji ogólna wydajność systemu nie spełniała wymogów aplikacji intensywnie korzystających z podsystemu graficznego. Oczywistym rozwiązaniem było wyeliminowanie dodatkowych wątków i koniecznych przełączeń kontekstów, poprzez przeniesienie systemu grafiki i okien do trybu jądra. Także po wywołaniu menedżera okien i funkcji GDI przez aplikację procedury te mają dostęp do innych komponentów wykonawczych Windows 2000, bez konieczności zmiany trybów pracy. Taki bezpośredni dostęp jest istotny zwłaszcza w przypadku, gdy GDI komunikuje się ze sterownikami video — jest to proces wymagający współpracy ze urządzeniami graficznymi z dużą częstotliwością i w szerokim paśmie.

Czy przez to, że USER i GDI działają w trybie jądra, Windows 2000 jest mniej stabilny?

Niektórzy zastanawiają się, czy przeniesienie tej ilości kodu do trybu jądra w znaczący sposób wpływa na stabilność systemu. Powodem, dla którego wpływ na stabilność systemu jest minimalny, jest fakt, że przed systemem Windows NT 4 (ale tak jest także obecnie), błąd (na przykład błąd dostępu) w procesie podsystemu Win32 trybu użytkownika (*Csrss.exe*) powodował załamanie systemu. To załamanie następowało, ponieważ proces nadrzędny względem *Csrss* (menedżer sesji *Smss*) wykonuje operację oczekiwania na uchwycie procesu *Csrss*, a jeśli to oczekiwanie się zakończy, zamyka system — dzieje się tak, ponieważ proces podsystemu Win32 był (i wciąż jest) krytyczny dla działania systemu. Ponieważ był to proces zawierający struktury danych opisujące wyświetlane okna, śmierć tego procesu pociągałaby za sobą załamanie interfejsu użytkownika. Jednakże nawet system Windows 2000 pełniący rolę serwera, bez procesów interaktywnych, nie może działać bez tego podsystemu, gdyż procesy serwera mogą korzystać z komunikatów okienkowych do kontroli wewnętrznego stanu aplikacji. W Windows 2000 błąd dostępu w tym samym kodzie, teraz działającym w trybie jądra, po prostu załamuje system szybciej, jako że wyjątki w trybie jądra powodują, że system pada.

Istnieje jednak teoretycznie dodatkowe niebezpieczeństwo, którego nie było przed przeniesieniem systemu grafiki i okien do trybu jądra. Ponieważ kod ten działa teraz w trybie jądra, błąd (taki jak niewłaściwy wskaźnik) mógłby powodować zakłócenie integralności chronionych struktur danych trybu jądra. Przed Windows NT 4 takie odwołania powodowałyby błąd dostępu, ponieważ niemożliwy jest zapis do stron jądra z trybu użytkownika. Ale i tak powodowałoby to załamanie systemu, jak to opisano wcześniej. Gdy kod działa teraz w trybie jądra, zapis pod błędny adres może nie powodować załamania systemu od razu, jeśli jednak zakłócona została spójność jakiejś struktury danych, system prawdopodobnie padnie niedługo później. Istnieje jednak małe prawdopodobieństwo, że takie odwołanie zniszczy zawartość jakiegoś bufora danych (a nie struktury danych), co powodowałoby w rezultacie zwrócenie błędnych danych do programu użytkownika lub zapis błędnych danych na dysk.

Inne niebezpieczeństwo może być spowodowane przeniesieniem w tryb jądra sterowników graficznych. Przedtem pewne części sterownika graficznego działały w obrębie *Csrss*, a pozostałe w trybie jądra. Teraz cały sterownik działa w trybie jądra. Mimo że Microsoft nie opracowuje wszystkich sterowników graficznych dla Windows 2000, to jednak współpracuje z producentami sprzętu, wspierając produkcję niezawodnych i wydajnych sterowników. Wszystkie sterowniki dostarczane z systemem są poddawane równie rygorystycznym testom, co pozostałe komponenty.

Wreszcie ważne jest, by pojąć, że ten projekt (podsystemy grafiki i okien działające w trybie jądra) nie jest zasadniczo ryzykowny. Jest identyczny z podejściem stosowanym w wielu innych sterownikach urządzeń (na przykład sterownikach kart sieciowych czy twardych dysków). Od początku istnienia Windows NT sterowniki te funkcjonowały stabilnie w trybie jądra.

Niektórzy spekulowali, że przeniesienie menedżera okien i modułu GDI do trybu jądra wpłynie negatywnie na wielozadaniowość z wywłaszczeniem w Windows 2000. Według tej teorii dodatkowy czas poświęcony na przetwarzanie przez podsystemWin32

w trybie jądra powodowałby, że inne wątki miałyby mniej czasu na działanie. Pogląd ten był spowodowany niezrozumieniem architektury Windows 2000. Prawdą jest, że w wielu nominalnie wywłaszczających systemach operacyjnych przetwarzanie w trybie jądra nigdy nie jest przerywane przez mechanizm szeregowania — lub jest przerywane tylko w pewnej ograniczonej liczbie momentów do tego przewidzianych. Jednakże w Windows 2000 wątki działające w obrębie centrum wykonawczego są wywłaszczane i szeregowane na równi z wątkami trybu użytkownika, zaś cały kod wykonawczy systemu może być wywłaszczany. Dzieje się tak, aby można było zapewnić wysoki stopień skalowalności na maszynach symetrycznego wieloprzetwarzania (SMP).

Inne spekulacje zawierały pogląd, że ta zmiana miałaby wpływ na skalowalność systemów SMP. Teoria ta mówiła: poprzednio interakcja miedzy aplikacją a menedżerem okien lub modułem GDI korzystała z dwóch wątków, jednego w aplikacji oraz jednego w *Csrss.exe*. Z tego powodu w systemie SMP te wątki mogą wykonywać się równolegle, zwiększając przepustowość. Taka analiza dowodzi nieznajomości technologii NT przed wprowadzeniem Windows NT 4. W większości przypadków wywołania aplikacji klienta kierowane do podsystemu Win32 odbywają się synchronicznie. Oznacza to, że wątek klienta jest blokowany w oczekiwaniu na wątek serwera i wznawia działanie dopiero po zakończeniu procedury przez wątek serwera. Dlatego nie można osiągnąć w ten sposób zrównoleglenia na maszynie SMP. Zjawisko to można łatwo zaobserwować w przypadku działającej aplikacji graficznej, korzystając z narzędzia *Wydajność* na maszynie SMP. Obserwator zauważy, że w komputerze dwuprocesorowym każdy procesor jest obciążony w około 50 procentach i względnie łatwo jest zlokalizować pojedynczy wątek *Csrss* związany z wątkiem działającej aplikacji. Co więcej, ponieważ dwa wątki są ze sobą ściśle związane (dotyczy to także ich stanu) pamięci podręczne procesorów muszą być stale opróżniane celem zapewnienia spójności. To ciągłe opróżnianie jest powodem, dla którego w Windows NT 3.51 jednowątkowa aplikacja graficzna z reguły działa nieco wolniej w systemie SMP niż na maszynie jednoprocesorowej.

W rezultacie zmiany w Windows NT 4 zwiększyły na maszynach SMP wydajność aplikacji intensywnie korzystających z funkcji okien oraz GDI, zwłaszcza gdy więcej niż jeden wątek aplikacji jest aktywny. Gdy dwa wątki są aktywne na dwuprocesorowej maszynie działającej pod kontrolą Windows NT 3.51, aż cztery wątki (dwa w aplikacji plus dwa w *Csrss*) rywalizują o czas dwóch procesorów. Mimo że z reguły tylko dwa są gotowe do pracy jednocześnie, brak spójnej sekwencji wykonywania wątków powoduje utratę lokalności odwołań i spójności pamięci podręcznych procesorów. Ta utrata wynika z faktu, że aktywne wątki wykonują się raz na jednym, raz na drugim procesorze. W architekturze Windows NT 4 każdy z dwóch wątków aplikacji zasadniczo posiada własny procesor. Zaś automatyczna afiniczność wątków w Windows 2000 powoduje, że jeden wątek jest wykonywany na jednym procesorze, w ten sposób zapewniając lokalność odwołań i zmniejszając potrzebę synchronizacji prywatnych pamięci podręcznych procesorów.

Podsumujmy: przeniesienie menedżera okien oraz GDI z trybu użytkownika do trybu jądra zapewniło większą wydajność bez znaczącego spadku stabilności i niezawodności systemu.

Tak więc, co zawierają procesy podsystemu Win32 działające w trybie użytkownika? Są odpowiedzialne za rysowanie i uaktualnianie okien konsoli, czyli tekstowych, ponieważ aplikacje konsolowe nie zlecają samodzielnie odrysowywania okna. Łatwo zaobserwować tę czynność — wystarczy otworzyć okno wiersza poleceń i przenieść ponad nim inne okno, co spowoduje wzmożoną aktywność procesu podsystemu Win32, odrysowującego okno konsoli. Ale poza obsługą okien konsoli, już tylko nieliczne funkcje Win32 powodują wysłanie komunikatu do procesu podsystemu Win32: tworzenie i niszczenie wątków i procesów, odwzorowanie liter napędów sieciowych oraz tworzenie plików tymczasowych. W ogólności działająca aplikacja Win32 nie powoduje wielu, o ile w ogóle, przejść do procesu podsystemu Win32.

Podsystem POSIX

POSIX, akronim z grubsza definiowany jako „przenośny interfejs systemów operacyjnych, oparty na systemie Uniks", oznacza zestaw międzynarodowych standardów uniksopodobnych interfejsów systemów operacyjnych. Standardy POSIX mają zachęcać producentów implementujących takie interfejsy do utrzymania zgodności ze standardem, co pozwoli programistom na łatwiejsze przenoszenie aplikacji pomiędzy platformami.

Windows 2000 implementuje tylko jeden z wielu standardów POSIX, POSIX.1, formalnie znany jako ISO/IEC 9945-1:1990 lub standard POSIX IEEE 1003.1-1990. Standard ten został zaimplementowany głównie celem spełnienia postawionych w połowie lat 80-tych wymagań rządu USA, nakładających obowiązek zgodności ze standardem POSIX.1, jak to zostało opisane w Federalnej Normie Przetwarzania Informacji (ang. *Federal Information Processing Standards — FIPS*), nr 151-2, opracowanej przez Narodowy Instytut Norm i Technologii (ang. *National Institute of Standards and Technology*). Windows NT 3.5, 3.51 oraz 4 zostały formalnie przetestowane i uzyskały certyfikat zgodności z normą FIPS 151-2.

Ponieważ zgodność z POSIX.1 była jednym z obowiązujących założeń Windows 2000, system operacyjny został zaprojektowany tak, by zapewniona była obsługa wszystkich funkcji przewidzianych dla implementacji podsystemu POSIX.1 (jak na przykład funkcja `fork`, która jest zaimplementowana w centrum wykonawczym Windows 2000, czy obsługa twardych odnośników do plików (ang. *hardlinks*) w systemie plików Windows 2000). Jednakże, ponieważ standard POSIX.1 definiuje ograniczony zestaw usług (takich jak kontrola procesów, komunikacja międzyprocesowa, proste znakowe funkcje I/O, itd.), podsystem POSIX zawarty w Windows 2000 nie jest pełnym środowiskiem programistycznym. A ponieważ aplikacje w Windows 2000 nie mogą mieszać wywołań funkcji z różnych podsystemów, aplikacje POSIX są ograniczone do zestawu usług zdefiniowanych w standardzie POSIX.1. Ograniczenie to oznacza, że program w podsystemie POSIX w Windows 2000 nie może utworzyć wątku czy okna ani nie może korzystać ze zdalnych wywołań procedur (RPC) czy gniazd (ang. *sockets*).

Aby zredukować to ograniczenie, Microsoft opracował produkt o nazwie Interix, zawierający rozszerzone środowisko podsystemu POSIX, udostępniające niemal 2000 funkcji uniksowych oraz 300 uniksopodobnych narzędzi i programów. (Więcej informacji na temat Microsoft Interix znajduje się na stronie *www.microsoft.com/windows2000/server/evaluation/business/interix.asp*). Z tym rozszerzeniem przenoszenie aplikacji uniksowych do podsystemu POSIX jest znacznie łatwiejsze. Jednakże, ponieważ programy wciąż oznaczone są jako pliki wykonywalne POSIX, nie mogą korzystać z funkcji Win32.

Aby przenieść aplikację uniksową do Windows 2000 i móc skorzystać z funkcji Win32, można nabyć bibliotekę ułatwiającą takie przenoszenie, jak ta dołączona do produktu MKS NuTRACKER Professional, firmy Mortice Kern Systems Inc. (*www.mks.com*). Dzięki takiemu podejściu aplikacja uniksowa może być skompilowana i skonsolidowana jako wykonywalny plik Win32, a co za tym idzie, korzystać z funkcji Win32.

ĆWICZENIE: Obserwacja rozruchu podsystemu POSIX

Podsystem POSIX jest skonfigurowany tak, że uruchamia się przy pierwszym uruchomieniu programu POSIX-owego, tak więc jego rozruch można zaobserwować przez włączenie programu POSIX-owego, jak któryś z narzędzi POSIX-owych dołączonych do Windows 2000 Resource Kit. (Narzędzia te znajdują się w katalogu *\Apps\ POSIX* na płycie instalacyjnej — nie są one instalowane podczas instalacji). Aby zaobserwować start podsystemu POSIX, musisz wykonać następujące czynności:

1. Uruchom wiersz poleceń.
2. Wpisz polecenie `tlist /t` i sprawdź, czy podsystem POSIX nie jest aktywny (to znaczy, czy pod procesem *Smss.exe* nie ma procesu *Psxss.exe*).
3. Uruchom jedno z narzędzi POSIX-owych, znajdujących się na płycie Windows 2000 Resource Kit, jak na przykład *\Apps\POSIX\Ls.exe*.
4. Można zauważyć krótką przerwę, podczas której uruchamiany jest podsystem POSIX, po czym polecenie `Ls` wyświetla zawartość katalogu.
5. Wpisz `tlist /t` ponownie. Tym razem rzuca się w oczy obecność *Psxss.exe*, będącego potomkiem *Smss.exe*.
6. Uruchom *Ls.exe* po raz drugi — tym razem reakcja jest szybsza (gdyż podsystem POSIX jest już uruchomiony).
7. Wywołaj *Ls.exe*, ale przerwij wyświetlanie poprzez naciśnięcie *Ctrl+S*. W innym oknie wiersza poleceń uruchom `tlist /t` i zauważ, że program obsługi POSIX (*Posix.exe*) był procesem, który został utworzony z pierwszego wiersza poleceń, a on z kolei stworzył proces *Ls.exe*. Tekst wyjściowy powinien mieć postać podobną do poniższej:

```
System (2)
  smss.exe (23) ———————————— Menadżer sesji
    csrss.exe (31) ———————————— Podsystem Win32
    .
    .
    .
    psxss.exe (187) ———————————— Podsystem POSIX
explorer.exe (69) Program Manager
  CMD.EXE (93) Command Prompt - ls
    posix.exe (178) ———————————— Proces obsługi POSIX
      ls.exe (97) ———————————— Wykonywana aplikacja POSIX-owa
```

Więcej informacji na temat obsługi aplikacji POSIX i OS/2 w Windows 2000 znajduje się w rozdziale 6.

Do kompilacji i konsolidacji aplikacja POSIX w Windows 2000 potrzebuje POSIX-owych plików nagłówkowych i bibliotek, dostarczanych z pakietem Platform SDK. Pliki wykonywalne POSIX są łączone z biblioteką podsystemu POSIX, *Psxdll.dll*. Ponieważ domyślnie Windows 2000 jest skonfigurowany tak, by uruchamiać podsystem POSIX na żądanie, przy pierwszym wywołaniu aplikacji POSIX-owej musi zostać uruchomiony proces podsystemu POSIX (*Psxss.exe*). Pozostaje on aktywny aż do ponownego rozruchu systemu. (Zabicie procesu podsystemu POSIX powoduje niemożność uruchomienia aplikacji POSIX-owych, aż do restartu systemu). Plik wykonywalny POSIX nie jest uruchamiany bezpośrednio — zamiast tego uruchamiany jest specjalny program pomocniczy o nazwie *Posix.exe*, który z kolei tworzy proces potomny, wykonujący aplikację POSIX.

Więcej informacji na temat podsystemu POSIX oraz przenoszenia aplikacji uniksowych do Windows 2000 znajdziesz w bibliotece MSDN, pod hasłami POSIX i Unix.

Podsystem OS/2

Podsystem środowiskowy OS/2, podobnie, jak wbudowany podsystem POSIX, ma dość ograniczoną przydatność, gdyż zapewnia interfejs OS/2 1.2, przez co obsługuje jedynie 16-bitowe aplikacje tekstowe oraz programy korzystające z mechanizmu video I/O (VIO). Mimo że Microsoft rozprowadzał zastępczy podsystem powłoki OS/2 1.2 Presentation Manager dla Windows NT, to pakiet ten nie obsługiwał aplikacji wymagających OS/2 2.x (ani późniejszych) — ani nie jest dostępny w wersji dla Windows 2000).

Również, ponieważ Windows 2000 nie zezwala aplikacjom użytkownika na bezpośredni dostęp do sprzętu, nie są obsługiwane programy OS/2 zawierające uprzywilejowane segmenty I/O, które usiłują wykonać instrukcje IN/OUT (celem odwołania się do w aplikacje zawierające instrukcje CLI/STI — ale wszystkie inne aplikacje OS/2 w systemie, a także pozostałe wątki procesu OS/2, który wykonał instrukcję CLI, zostają zawieszone aż do wykonania instrukcji STI. Warta wspomnienia jest specjalna możliwość wywoływania 32-bitowych bibliotek DLL z poziomu 16-bitowych aplikacji OS/2 w Windows 2000, co może być użyteczne przy przenoszeniu programów.

16 MB ograniczenie pamięci z rzeczywistego OS/2 1.2 nie stosuje się do Windows 2000 — podsystem OS/2 korzysta z 32-bitowej wirtualnej przestrzeni adresowej Windows 2000 i zapewnia aplikacjom OS/2 1.2 do 512 MB pamięci. Ilustruje to rysunek 2.5.

Obszar segmentowany składa się na 512 MB wirtualnej przestrzeni adresowej, która jest zarezerwowana na początku i zajmowana lub zwalniana, gdy aplikacje 16-bitowe potrzebują pamięci. Podsystem OS/2 prowadzi lokalną tablicę deskryptorów (LDT) dla każdego procesu, a współdzielone segmenty pamięci znajdują się na tych samych pozycjach tablic LDT wszystkich procesów OS/2.

Jak to zostanie omówione w rozdziale 6., wątki są elementami wykonywanego programu, przez co muszą uczestniczyć podczas szeregowania zadań w rywalizacji o czas procesora. Mimo że przedział priorytetów w Windows 2000 rozciąga się od 0 do 31, 64 poziomy priorytetów systemu OS/2 (0 do 63) są odwzorowywane na dynamiczne priorytety Windows 2000 o numerach od 1 do 15. Wątki OS/2 nigdy nie otrzymują priorytetów czasu rzeczywistego Windows 2000, posiadających numery od 16 do 31.

Rysunek 2.5. *Rozkład pamięci wirtualnej podsystemu OS/2*

Tak jak w przypadku podsystemu POSIX podsystem OS/2 jest uruchamiany automatycznie przy pierwszym wywołaniu pliku wykonywalnego OS/2. Pozostaje aktywny aż do ponownego rozruchu systemu.

Więcej informacji na temat obsługi aplikacji POSIX i OS/2 przez Windows 2000 znajduje się w rozdziale 6.

Ntdll.dll

Ntdll.dll jest specjalną biblioteką wspierającą system, wspomagającą głównie korzystanie z bibliotek podsystemowych. Zawiera dwa rodzaje funkcji:

- Procedury przekazujące wywołania usług systemowych do centrum wykonawczego Windows 2000.
- Wewnętrzne funkcje pomocnicze, wykorzystywane przez podsystemy, biblioteki podsystemów i inne komponenty.

Pierwsza grupa funkcji zapewnia dostępny z trybu użytkownika interfejs do wewnętrznych usług systemowych Windows 2000. Istnieje ponad 200 takich usług, na przykład `NtCreateFile`, `NtSetEvent`, itd. Jak już wspomniano wcześniej, wiele z tych funkcji jest dostępnych za pośrednictwem Win32 API. (Kilka z nich jednak nie jest dostępnych i jest na wewnętrzne potrzeby Microsoftu).

Dla każdej z tych funkcji *Ntdll* posiada punkt wejściowy o tej samej nazwie. Kod funkcji zawiera zależne od architektury instrukcje powodujące przejście do trybu jądra celem wywołania dyspozytora usług systemowych (omówionego bardziej szczegółowo

w rozdziale 3.), który po weryfikacji parametrów wywołuje z *Ntoskrnl.exe* właściwą usługę systemową trybu jądra. Usługa ta zawiera prawdziwy kod.

Ntdll zawiera również szereg funkcji pomocniczych, jak moduł uruchamiający programy (funkcje, których nazwy rozpoczynają się od `Ldr`), menedżer sterty czy funkcje komunikacji procesu podsystemu Win32 (funkcje, których nazwy rozpoczynają się od `Csr`), jak również ogólne funkcje biblioteczne (funkcje, których nazwy rozpoczynają się od `Rtl`). Zawiera również moduł dyspozytora asynchronicznych wywołań procedur (ang. *asynchronous procedure call — APC*) trybu użytkownika oraz dyspozytora wyjątków (APC oraz wyjątki zostaną omówione w rozdziale 3.).

Centrum wykonawcze

Centrum wykonawcze Windows 2000 jest górną warstwą *Ntoskrnl.exe*. (Dolną warstwą jest jądro). Centrum wykonawcze zawiera następujące rodzaje funkcji:

- Funkcje opublikowane i dostępne z trybu użytkownika. Funkcje te nazywają się *usługami systemowymi* i są udostępniane za pośrednictwem *Ntdll*. Większość z tych usług jest dostępna poprzez Win32 API lub interfejsy innych podsystemów środowiskowych. Niektóre usługi nie są jednak dostępne za pośrednictwem żadnej udokumentowanej funkcji podsystemowej. (Do przykładów można zaliczyć LPC oraz różne funkcje zapytań, jak `NtQueryInformationxxx`, funkcje specjalizowane, jak `NtCreatePagingFile`, itd.)
- Funkcje dostępne jedynie z trybu jądra, opublikowane i udokumentowane w pakietach Windows 2000 DDK oraz Windows 2000 Installable File System (IFS) Kit. (Informacje na temat pakietu Windows 2000 IFS Kit znajdują się na stronie *www.microsoft.com/ddk/ifskit*).
- Funkcje udostępniane i dostępne z trybu jądra, ale nie udokumentowane w pakietach Windows 2000 DDK oraz IFS Kit (jak na przykład funkcje wywoływane przez sterownik video w czasie rozruchu systemu — nazwy tych funkcji rozpoczynają się od `Inbv`).
- Funkcje, które są zdefiniowane jako symbole globalne, ale nie są udostępniane. Do nich należą wewnętrzne funkcje pomocnicze, wywoływane z poziomu *Ntoskrnl*, jak funkcje o nazwach rozpoczynających się od `Iop` (wewnętrzne funkcje pomocnicze menedżera operacji I/O) lub `Mi` (wewnętrzne funkcje pomocnicze systemu zarządzania pamięcią).
- Funkcje określone jako wewnętrzne w obrębie jakiegoś modułu, które nie są symbolami globalnymi.

Centrum wykonawcze zawiera następujące istotne komponenty, z których każdy jest omówiony szczegółowo w kolejnych rozdziałach tej książki:

- *Menedżer konfiguracji* (omówiony w rozdziale 5.) odpowiada za implementację i zarządzanie rejestrem systemowym.
- *Menedżer procesów i wątków* (omówiony w rozdziale 6.) tworzy oraz niszczy procesy i wątki. Funkcje obsługujące procesy i wątki zaimplementowane są w jądrze Windows 2000. Centrum wykonawcze nadaje dodatkową semantykę i funkcjonalność tym niskopoziomowym obiektom.

- *Monitor bezpieczeństwa odwołań* (opisany w rozdziale 8.) wymusza politykę bezpieczeństwa na lokalnym komputerze. Strzeże zasobów systemu operacyjnego, prowadząc na bieżąco kontrolę i ochronę obiektów.
- *Menedżer I/O* (opisany w rozdziale 9.) implementuje niezależne od sprzętu operacje wejścia-wyjścia i jest odpowiedzialny za przekierowanie żądań do odpowiednich sterowników urządzeń, gdzie zostaną przetworzone.
- *Menedżer Plug and Play* (omówiony w rozdziale 9.) określa, które sterowniki są potrzebne do obsługi konkretnego urządzenia i ładuje te sterowniki. Pobiera wymagania zasobów sprzętowych każdego urządzenia. Na podstawie wymagań sprzętowych tych urządzeń menedżer PnP przyznaje odpowiednie zasoby, jak porty I/O, przerwania IRQ, kanały DMA oraz adresy pamięci. Jest również odpowiedzialny za wysyłanie odpowiednich komunikatów w przypadku zmiany stanu urządzeń (jak usunięcie czy dodanie urządzenia) w systemie.
- *Menedżer energii* (omówiony w rozdziale 9.) koordynuje zdarzenia związane z poborem mocy przez system i generuje komunikaty dotyczące zarządzania energią, wysyłane do sterowników urządzeń. Gdy system nie wykonuje obliczeń, konfiguracja menedżera energii może przewidywać obniżenie poboru mocy poprzez uśpienie procesora. Zmiany w poborze mocy przez poszczególne urządzenia są obsługiwane przez sterowniki urządzeń, ale koordynowane przez menedżera energii.
- *Procedury instrumentacji zarządzania Windows WDM* (ang. *WDM Windows Management Instrumentation*, omówione w rozdziale 5.) pozwalają sterownikom urządzeń na publikację informacji na temat wydajności i konfiguracji, a także na odbieranie poleceń usług WMI z trybu użytkownika. Odbiorcy informacji WMI mogą znajdować się na maszynie lokalnej lub zdalnej, podłączonej do sieci.
- *Menedżer pamięci podręcznej* (omówiony w rozdziale 11.) zwiększa wydajność plikowych operacji I/O poprzez składowanie w pamięci danych dyskowych, do których dostęp odbywał się niedawno (a także przez opóźnianie operacji zapisu, poprzez składowanie uaktualnień w pamięci przed wysłaniem ich na dysk). Jak się okaże, wykorzystywany jest tutaj mechanizm odwzorowania plików, zaimplementowany przez system zarządzania pamięcią.
- *Menedżer pamięci wirtualnej* (opisany w rozdziale 7.) implementuje *pamięć wirtualną*, mechanizm zarządzania pamięcią zapewniający każdemu procesowi dużą, prywatną przestrzeń adresową, która może przekraczać dostępną pamięć fizyczną. Menedżer pamięci zapewnia również niskopoziomowe funkcje pomocnicze dla menedżera pamięci podręcznej.

Dodatkowo, centrum wykonawcze zawiera cztery główne grupy funkcji pomocniczych, wykorzystywane przez wymienione komponenty. Około jednej trzeciej tych funkcji jest udokumentowanych w DDK, ponieważ korzystają z nich również sterowniki urządzeń. Są cztery kategorie funkcji pomocniczych:

- *Menedżer obiektów*, który tworzy, zarządza i niszczy obiekty Windows 2000 oraz abstrakcyjne typy danych wykorzystywane do reprezentacji zasobów systemu operacyjnego takich jak procesy, wątki i różne obiekty synchronizacyjne. Menedżer obiektów zostanie omówiony w rozdziale 3.

- *Mechanizm LPC* (omówiony w rozdziale 3.) przekazuje komunikaty między procesem klienta a procesem serwera na tym samym komputerze. LPC jest elastyczną, zoptymalizowaną wersją zdalnych wywołań procedur (ang. *remote procedure call — RPC*), przemysłowego standardu komunikacji procesów klientów i serwerów przez sieć.
- Szeroki wachlarz zwykłych funkcji *bibliotecznych*, takich jak przetwarzanie łańcuchów znakowych, operacje arytmetyczne, konwersja typów danych oraz przetwarzanie struktur bezpieczeństwa.
- *Procedury pomocnicze centrum wykonawczego*, jak przydział pamięci (pule stronicowana i niestronicowana), dostęp do pamięci, jak również dwa specjalne typy obiektów synchronizacyjnych: zasoby i szybkie muteksy.

Jądro

Jądro składa się z zestawu funkcji zawartych w *Ntoskrnl.exe*, zapewniających podstawowe mechanizmy (jak szeregowanie wątków i usługi synchronizacyjne) wykorzystywane przez komponenty centrum wykonawczego, jak również niskopoziomowe funkcje, zależne od platformy sprzętowej (jak obsługa przerwań i wyjątków), które różnią się między różnymi architekturami procesorów. Kod jądra został napisany głównie w C, ze wstawkami asemblerowymi realizującymi zadania wymagające dostępu do specjalizowanych instrukcji procesora oraz rejestrów niedostępnych w łatwy sposób z poziomu języka C.

Tak, jak w przypadku różnych funkcji pomocniczych centrum wykonawczego, wymienionych w poprzednim podrozdziale, niektóre z funkcji jądra jest udokumentowanych w DDK (są to funkcje o nazwach rozpoczynających się od `Ke`), ponieważ są one konieczne w implementacji sterowników urządzeń.

Obiekty jądra

Jądro zapewnia niskopoziomowe fundamenty dla dobrze określonych, przewidywalnych podstawowych mechanizmów systemu operacyjnego, pozwalających wysokopoziomowym komponentom centrum wykonawczego wykonywać przypisane im zadania. Jądro oddziela się od reszty centrum wykonawczego, skupiając się na implementacji mechanizmów systemowych i unikając podejmowania decyzji dotyczących polityki systemu. Decyzje takie jądro pozostawia w gestii centrum wykonawczego, z wyjątkiem szeregowania wątków, które implementuje.

Poza jądrem centrum wykonawcze reprezentuje wątki i inne zasoby współdzielone jako obiekty. Obiekty te wymagają dodatkowego nakładu związanego z polityką systemu, na przykład wymagają uchwytów do manipulacji nimi, kontroli bezpieczeństwa do ochrony czy zmniejszania limitów dostępnych zasobów podczas tworzenia obiektów. Ten nakład jest eliminowany w jądrze, które implementuje zestaw prostych obiektów, zwanych *obiektami jądra*. Obiekty te wspomagają kontrolę centralnego przetwarzania i pozwalają na tworzenie obiektów centrum wykonawczego. Wiele z obiektów centrum wykonawczego zawiera w sobie jeden lub więcej obiekt jądra, włącznie z określonymi przez jądro atrybutami.

Jeden zestaw obiektów jądra, zwany *obiektami kontrolnymi*, ustanawia semantykę kontroli różnych funkcji systemu operacyjnego. W skład tego zestawu wchodzi obiekt APC, obiekt *opóźnionego wywoływania procedury* (ang. *deferred procedure call — DPC*) oraz kilka obiektów używanych przez menedżera I/O, jak obiekt przerwania.

Inny zestaw obiektów jądra, zwanych *obiektami dyspozycyjnymi*, zapewnia mechanizmy synchronizacyjne, mające wpływ na szeregowanie wątków. Do obiektów dyspozycyjnych należą wątek jądra, mutex (wewnętrznie nazywany *mutantem*), zdarzenie, para zdarzeń jądra, semafor, stoper oraz stoper z możliwością oczekiwania. Centrum wykonawcze korzysta z funkcji jądra przy tworzeniu instancji obiektów jądra, przy operowaniu nimi oraz przy konstrukcji bardziej złożonych obiektów dostępnych z trybu użytkownika. Obiekty są omówione bardziej szczegółowo w rozdziale 3., zaś procesy i wątki są opisane w rozdziale 6.

Obsluga sprzętu

Innym ważnym zadaniem jądra jest odizolowanie centrum wykonawczego oraz sterowników urządzeń od zróżnicowania platform sprzętowych obsługiwanych przez Windows 2000. Zadanie to oznacza uwzględnienie różnic w implementacji takich funkcji, jak obsługa przerwań i wyjątków czy synchronizacja wieloprocesorowa.

Nawet w przypadku funkcji do tego stopnia zależnych od sprzętu projekt jądra usiłuje do maksimum zwiększyć ilość wspólnego kodu. Jądro obsługuje zestaw interfejsów, które są przenośne i semantycznie identyczne w różnych architekturach. Większość kodu implementującego ten przenośny interfejs również jest identyczna dla różnych architektur.

Niektóre z tych interfejsów są jednak implementowane w różny sposób na różnych platformach sprzętowych. Z kolei niektóre z tych interfejsów są częściowo zaimplementowane w postaci kodu dla konkretnej platformy. Te niezależne od architektury interfejsy są dostępne na każdej maszynie, a semantyka interfejsu będzie taka sama, niezależnie od tego, czy kod na tę konkretną platformę czymś się różni. Niektóre interfejsy jądra (jak procedury `spinlock`, omówione w rozdziale 3.) są w rzeczywistości zaimplementowane w warstwie HAL (omówionej w następnym podrozdziale), ponieważ ich implementacja może się różnić w różnych systemach należących do tej samej architektury.

Jądro zawiera również niewielki fragment kodu ze specyficznymi dla platformy x86 interfejsami, koniecznymi do obsługi starych programów dla MS-DOS-a. Te interfejsy nie są przenośne, w tym sensie, że nie są dostępne na żadnej innej platformie — nie będą obecne. Ten specyficzny dla x86 kod zapewnia na przykład możliwość operacji na globalnych i lokalnych tablicach deskryptorów (GDT i LDT), będących sprzętowymi mechanizmami procesorów rodziny x86.

Do innych przykładów kodu specyficznego dla konkretnej architektury można zaliczyć interfejs obsługujący bufor translacji i pamięć podręczną procesora. Obsługa ta wymaga innego kodu dla różnych architektur, co jest spowodowane sposobem implementacji tych mechanizmów w procesorach.

Innym przykładem jest przełączanie kontekstów. Mimo że na wysokim poziomie stosowany jest ten sam algorytm selekcji wątków i przełączania kontekstu (kontekst poprzedniego wątku jest zapisywany, ładowany jest kontekst nowego wątku, po czym uruchamiany jest nowy wątek), to istnieją różnice implementacji tego mechanizmu w różnych procesorach. Ponieważ kontekst jest określony przez stan procesora (stan rejestrów, itd.), zapisywane i ładowane dane zależą od architektury.

Warstwa uniezależnienia od sprzętu

Jak już wspomniano na początku rozdziału, jednym z kluczowych założeń projektu Windows 2000 była przenośność na różne platformy sprzętowe. Warstwa uniezależnienia od sprzętu (ang. *hardware abstraction layer — HAL*) odgrywa zasadniczą rolę w zapewnieniu tej przenośności. HAL jest zewnętrznym, pracującym w trybie jądra modułem (*Hal.dll*). Zapewnia on niskopoziomowy interfejs do platformy sprzętowej, na której działa Windows 2000. Jego zadaniem jest ukrycie wszystkich szczegółów sprzętowych, jak interfejsy I/O, kontrolery przerwań i mechanizmy komunikacji międzyprocesorowej — funkcji, które są specyficzne dla danej architektury bądź zależne od maszyny.

Tak więc, zamiast odwoływać się do sprzętu bezpośrednio, wewnętrzne komponenty Windows 2000, jak również sterowniki urządzeń stworzone przez niezależnych producentów, chcąc pobrać informacje o platformie wywołują procedury warstwy HAL, co zapewnia przenośność. Z tego powodu procedury warstwy HAL są udokumentowane w pakiecie Windows 2000 DDK. Więcej informacji na temat warstwy HAL i jej zastosowania przez sterowniki urządzeń znajduje się w DDK.

Mimo że płyta instalacyjna Windows 2000 zawiera kilka wersji modułu HAL (wymienionych w tabeli 2.5), tylko jedna z nich jest wybierana podczas instalacji i kopiowana na dysk komputera pod nazwą *Hal.dll*. (Inne systemy operacyjne, jak VMS, dokonują wyboru funkcjonalnego odpowiednika modułu HAL podczas rozruchu systemu.) Tak więc, nie można założyć, że system zainstalowany na dysku uruchomi się na innym procesorze, jeśli do obsługi tego procesora wymagany jest inny moduł HAL.

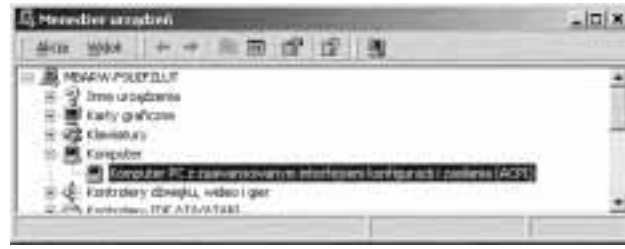
Tabela 2.5. *Lista wersji warstwy HAL*

Nazwa pliku HAL	Obsługiwane systemy
Hal.dll	Standardowe komputery PC
Halacpi.dll	Komputery PC wyposażone w zaawansowany interfejs konfiguracji i zarządzania energią (ang. *Advanced Configuration and Power Interface — ACPI*)
Halapic.dll	Komputery PC wyposażone w zaawansowany programowalny kontroler przerwań (ang. *Advanced Programmable Interrupt Controller — APIC*)
Halaacpi.dll	Komputery PC wyposażone w APIC i ACPI
Halmps.dll	Wieloprocesorowe komputery PC
Halmacpi.dll	Wieloprocesorowe komputery PC wyposażone w ACPI
Halborg.dll	Stacje robocze Silicon Graphics (już nie produkowane)
Halsp.dll	Compaq SystemPro

ĆWICZENIE: Identyfikacja wersji uruchomionej warstwy HAL

Istnieją dwa sposoby określenia wersji uruchomionej warstwy HAL:

1. Otwórz plik *\Winnt\Repair\Setup.log*, znajdź *Hal.dll* i spójrz na nazwę pliku po znaku równości. Jest to nazwa pliku HAL pobranego z pliku *Driver.cab* na płycie dystrybucyjnej.
2. W *Menedżerze urządzeń* (naciśnij prawym klawiszem myszy na ikonie *Mój komputer* na pulpicie, wybierz *Właściwości*, wybierz zakładkę *Sprzęt* i wybierz *Menedżera urządzeń*), poszukaj nazwy „sterownika" urządzenia o nazwie *Komputer*. Na przykład poniższy ekran pochodzi z systemu w którym działa wersja ACPI HAL:



Sterowniki urządzeń

Mimo że sterowniki urządzeń zostaną szczegółowo omówione w rozdziale 9., ta część zawiera krótki opis rodzajów sterowników i wyjaśnia, jak zidentyfikować sterowniki zainstalowane i załadowane w systemie.

Sterowniki urządzeń są zewnętrznymi modułami trybu jądra (o plikach z reguły noszących rozszerzenie *.sys*), które służą jako interfejs pomiędzy menedżerem I/O a odpowiednimi urządzeniami. Działają one w trybie jądra, w jednym z trzech kontekstów:

- W kontekście wątku użytkownika, który zainicjował funkcję I/O.
- W kontekście systemowego wątku trybu jądra.
- Jako rezultat przerwania (czyli poza kontekstem konkretnego procesu czy wątku — niezależnie od tego, który z nich był aktywny w momencie nadejścia przerwania).

Jak to zostało powiedziane w poprzednim podrozdziale, sterowniki urządzeń w Windows 2000 nie wykonują operacji na sprzęcie bezpośrednio, tylko wywołują funkcje warstwy HAL, komunikujące się z urządzeniami. Sterowniki z reguły pisane są w C (czasem w C++) przez co, zakładając prawidłowe wykorzystanie funkcji modułu HAL, ich kod źródłowy może być przenośny pomiędzy różnymi architekturami wspieranymi przez Windows 2000, natomiast ich kod binarny jest przenośny w obrębie jednej architektury.

Istnieje kilka rodzajów sterowników urządzeń:

- *Sterowniki urządzeń fizycznych* odpowiadają za operacje na sprzęcie (za pośrednictwem warstwy HAL), pozwalające na wysyłanie i odbiór danych z fizycznych urządzeń lub sieci. Istnieje wiele rodzajów sterowników urządzeń fizycznych, jak sterowniki magistral, sterowniki urządzeń kontaktu z użytkownikiem, sterowniki pamięci masowej, itd.
- *Sterowniki systemów plików* są sterownikami Windows 2000 które odbierają żądania związane z plikowymi operacjami I/O i tłumaczą je na żądania kierowane do konkretnych urządzeń.
- *Filtrujące sterowniki systemów plików*, jak na przykład oprogramowanie wykonujące lustrzane kopie i szyfrowanie dysków, przechwytują operacje I/O i wykonują dodatkowe przetwarzanie przed przekazaniem danych do następnej warstwy.
- *Sterowniki przekierowania sieciowego* oraz *serwery* są sterownikami systemów plikowych, które transmitują żądania I/O związane z systemem plików do maszyny podłączonej do sieci, i, odpowiednio, odbierają takie żądania.
- *Sterowniki protokołów* implementują protokoły sieciowe, takie jak TCP/IP, NetBEUI czy IPX/SPX.
- *Strumieniowe filtrujące sterowniki jądra* są połączone w łańcuch dokonujący przetwarzania sygnału zawartego w strumieniu danych, jak na przykład nagrywanie lub odtwarzanie dźwięku czy obrazu.

Ponieważ zainstalowanie sterownika urządzenia jest jedynym sposobem dodania do systemu wykonywanego w trybie jądra kodu stworzonego przez użytkownika, niektórzy programiści wykorzystują sterowniki urządzeń jako sposób dostępu do wewnętrznych funkcji lub struktur danych systemu operacyjnego, niedostępnych z trybu użytkownika (ale udokumentowanych i obsługiwanych przez DDK). Na przykład w skład wielu z narzędzi dostępnych na *www.sysinternals.com* wchodzi aplikacja Win32 GUI i sterownik służący do pobierania informacji o wewnętrznym stanie systemu, niedostępnych z poziomu Win32 API.

Rozszerzenia sterowników urządzeń Windows 2000

Windows 2000 obsługuje standard Plug and Play, Power Options oraz rozszerzony model sterowników Windows NT, zwany modelem sterowników Windows (ang. *Windows Device Model — WDM*). Windows 2000 potrafi uruchomić istniejące sterowniki Windows NT 4, ale ponieważ nie obsługują one standardów Plug and Play, ani Power Options, system korzystający z tych sterowników będzie uboższy o te funkcje.

Z perspektywy WDM istnieją trzy typy sterowników:

- *Sterownik magistrali* obsługuje kontroler magistrali, mostek lub dowolne urządzenie posiadające urządzenia mu podległe. Sterowniki magistrali są wymaganymi komponentami, w ogólności dostarczanymi przez Microsoft. Każdy rodzaj magistrali (jak PCI, PCMCIA czy USB) w systemie posiada jeden sterownik magistrali. Niezależni producenci mogą dostarczać sterowniki dla nowych magistral, jak VMEbus, Multibus czy Futurebus.

- *Sterownik funkcji* jest głównym sterownikiem i zapewnia operacyjny interfejs dla swojego urządzenia. Jest sterownikiem wymaganym, chyba że urządzenie jest używane niskopoziomowo (w takiej implementacji operacje wejścia-wyjścia są dokonywane przez sterownik magistrali i dowolne sterowniki filtrujące, jak *SCSI PassThru*). Sterownik funkcji jest z definicji sterownikiem, który wie najwięcej o konkretnym urządzeniu i jest z reguły jedynym sterownikiem, który odwołuje się do rejestrów urządzenia.
- *Sterownik filtrujący* jest stosowany dla nadania urządzeniu (lub istniejącemu sterownikowi) dodatkowej funkcjonalności lub dla modyfikacji żądań i odpowiedzi I/O pochodzących od innych sterowników (mechanizm ten jest często wykorzystywany dla zamaskowania błędów sprzętowych, powodujących, że urządzenie zgłasza błędne żądania dotyczące zasobów systemowych). Sterowniki filtrujące są opcjonalne i mogą występować w dowolnej liczbie, zainstalowane ponad lub pod sterownikiem funkcji lub ponad sterownikiem magistrali. Sterowniki filtrujące są z reguły dostarczane przez oryginalnych producentów sprzętu (OEM) lub niezależnych sprzedawców sprzętu (IHV).

W środowisku sterowników WDM nie występuje sytuacja, kiedy pojedynczy sterownik ma wpływ na wszystkie aspekty urządzenia: sterownik magistrali odpowiada za zgłaszanie obecności urządzeń na szynie menedżera PnP, podczas gdy sterownik funkcji zapewnia możliwość operacji na urządzeniu.

W większości przypadków niskopoziomowe sterowniki filtrujące modyfikują zachowanie obsługiwanych urządzeń. Na przykład, jeśli urządzenie zgłasza sterownikowi magistrali, że wymaga czterech portów I/O, podczas gdy w rzeczywistości potrzebuje 16 portów, niskopoziomowy sterownik filtrujący może przechwycić listę zasobów sprzętowych przekazywaną przez sterownik magistrali do menedżera PnP i poprawić liczbę wymaganych portów.

Wysokopoziomowe sterowniki filtrujące z reguły zapewniają dodatkowe funkcje urządzenia. Na przykład wysokopoziomowy, filtrujący sterownik klawiatury może wymusić dodatkowe kontrole bezpieczeństwa.

Obsługa przerwań jest wyjaśniona w rozdziale 3. Dalsze szczegóły dotyczące menedżera I/O, WDM, Plug and Play oraz Power Options są omówione w rozdziale 9.

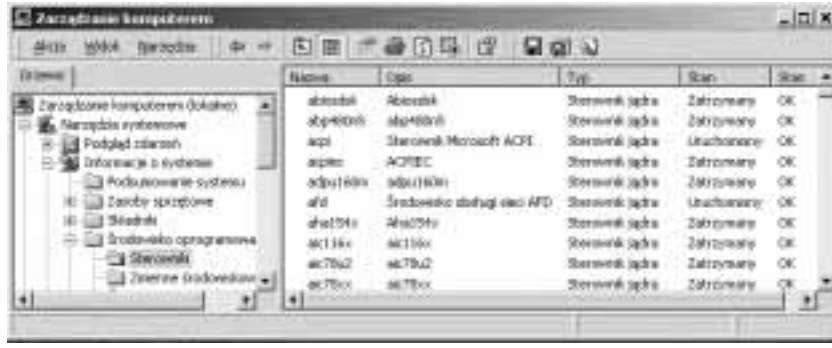
Podglądanie nieudokumentowanych interfejsów

Samo badanie nazw udostępnianych lub globalnych symboli w kluczowych plikach systemowych (jak *Ntoskrnl.exe*, *Hal.dll* czy *Ntdll.dll*) może być pouczające — można uzyskać ogólne pojęcie o możliwościach Windows 2000, tudzież o funkcjach na dzień dzisiejszy nie udokumentowanych, a obsługiwanych. Oczywiście, samo poznanie nazw tych funkcji nie oznacza, że można, czy powinno się je wywoływać — interfejsy te są nieudokumentowane i mogą ulegać zmianom. Poznanie nazw tych funkcji służy po prostu lepszemu zgłębieniu wewnętrznego działania Windows 2000, a nie ominięciu istniejących interfejsów.

Na przykład lista funkcji zawartych w *Ntdll.dll* jest równoważna liście wszystkich usług systemowych udostępnianych przez Windows 2000 bibliotekom podsystemów trybu użytkownika, co można zestawić ze zbiorem funkcji podsystemów. Mimo że wiele z tych

ĆWICZENIE: Przeglądanie zainstalowanych sterowników urządzeń

Listę zainstalowanych sterowników można obejrzeć dzięki narzędziu *Zarządzanie komputerem*. (Z menu *Start* wybierz *Programy/Narzędzia administracyjne*, a następnie *Zarządzanie komputerem*. Ewentualnie w *Panelu sterowania* możesz otworzyć *Narzędzia administracyjne* i wybrać *Zarządzanie komputerem*). W programie *Zarządzanie komputerem* rozwiń *Informacje o systemie* a następnie *Środowisko oprogramowania*, po czym otwórz *Sterowniki*. Poniżej znajduje się przykładowa lista zainstalowanych sterowników:



Okno to zawiera listę sterowników urządzeń zdefiniowanych w rejestrze, ich rodzaj oraz ich stan (*Uruchomiony* lub *Zatrzymany*). Sterowniki urządzeń oraz procesy usługowe Win32 są zdefiniowane w tym samym miejscu: *HKLM\SYSTEM\CurrentControlSet\Services*. Jednakże są one rozróżniane poprzez kod typu — typ 1 oznacza sterownik trybu jądra, zaś typ 2 oznacza sterownik systemu plikowego. Więcej szczegółów na temat zebranych w rejestrze informacji na temat sterowników urządzeń znajduje się w pliku dokumentacji technicznej Rejestru Windows 2000 (*Regentry.chm*) w Windows 2000 Resource Kit, pod kluczem *HKLM\SYSTEM\CurrentControlSet\Services*.

Oprócz tego listę aktualnie załadowanych sterowników można obejrzeć za pomocą programu *Drivers* (*Drivers.exe* w Windows 2000 Resource Kit) lub narzędzia *Pstat* (*Pstat.exe* z pakietu Platform SDK). Poniżej znajduje się fragment listy wyświetlonej przez program *Drivers*:

```
C:\>drivers
  ModuleName   Code   Data  Bss   Paged    Init          LinkDate
-------------------------------------------------------------------
ntoskrnl.exe 429184  96896   0  775360  138880  Tue Dec 07 18:41:11 1999
     hal.dll  25856   6016   0   16160   10240  Tue Nov 02 20:14:22 1999
 BOOTVID.DLL   5664   2464   0       0     320  Wed Nov 03 20:24:33 1999
    ACPI.sys  92096   8960   0   43488    4448  Wed Nov 10 20:06:04 1999
  WMILIB.SYS    512      0   0    1152     192  Sat Sep 25 14:36:47 1999
     pci.sys  12704   1530   0   31264    4608  Wed Oct 27 19:11:08 1999
  isapnp.sys  14368    832   0   22944    2048  Sat Oct 02 16:00:35 1999
compbatt.sys   2496      0   0    2880    1216  Fri Oct 22 18:32:49 1999
   BATTC.SYS    800      0   0    2976     704  Sun Oct 10 19:45:37 1999
intelide.sys   1760     32   0       0     128  Thu Oct 28 19:20:03 1999
```

```
 PCIIDEX.SYS   4544    480   0   10944    1632  Wed Oct 27 19:02:19 1999
  pcmcia.sys  32800   8864   0   23680    6240  Fri Oct 29 19:20:08 1999
  ftdisk.sys   4640     32   0   95072    3392  Mon Nov 22 14:36:23 1999
...
-------------------------------------------------------------
      Total 4363360  580320  0 3251424  432992
```

Na liście znajduje się każdy komponent trybu jądra (*Ntoskrnl*, HAL oraz sterowniki urządzeń), razem z rozmiarami sekcji każdego modułu.

Narzędzie *Pstat* również wyświetla listę załadowanych sterowników, ale poprzedza ją listą procesów i wątków każdego procesu. *Pstat* wyświetla jedną ważną informację, której nie uwzględnia program *Drivers*: adres pod którym rozpoczyna się w przestrzeni adresowej systemu dany komponent. Jak to zostanie wyjaśnione później, adres ten pełni kluczową rolę w odwzorowaniu uruchomionych wątków systemowych na ich macierzyste sterowniki urządzeń.

funkcji jest tożsamościowo odwzorowywanych przez udokumentowane i obsługiwane funkcje Win32, niektóre nie są dostępne przez Win32 API. (Więcej informacji na ten temat zawiera artykuł „Inside the Native API" z *www.sysinternals.com*, którego kopia znajduje się na płycie CD dołączonej do tej książki).

Co za tym idzie, interesujące wyniki daje również analiza funkcji wywoływanych przez biblioteki podsystemu Win32 (jak *Kernel32.dll* czy *Advapi32.dll*), a także wykorzystywanych przez nie procedur *Ntdll*.

Kolejnym interesującym plikiem jest *Ntoskrnl.exe* — mimo iż wiele spośród wykorzystywanych przez sterowniki urządzeń trybu jądra funkcji jest udokumentowanych w Windows 2000 DDK, to sporo procedur nie jest. Interesująca może być również tabela funkcji wywoływanych przez *Ntoskrnl* i HAL. Tabela ta zawiera listę funkcji warstwy HAL wywoływanych przez *Ntoskrnl* oraz vice versa.

Tabela 2.6 zawiera listę najczęściej używanych przedrostków nazw funkcji pochodzących z komponentów centrum wykonawczego.

Każdy z tych ważniejszych komponentów stosuje jeszcze wariacje nazwy celem odróżnienia funkcji wewnętrznych — wtedy jest to pierwsza litera przedrostka, po której następuje litera `i` (od słowa *internal — wewnętrzny*) lub do pełnego przedrostka dodana jest litera `p` (od *private — prywatny*). Na przykład `Ki` oznacza wewnętrzne funkcje jądra, zaś `Psp` odnosi się do wewnętrznych pomocniczych funkcji operujących na procesach.

Łatwiej odszyfrować nazwy tych eksportowanych funkcji, biorąc pod uwagę konwencję nazewnictwa procedur systemowych Windows 2000. Ogólny format jest następujący:

```
<Przedrostek><Operacja><Obiekt>
```

W tym formacie *Przedrostek* oznacza wewnętrzny komponent udostępniający tę funkcję. *Operacja* oznacza, co dzieje się z obiektem bądź zasobem, zaś *Obiekt* identyfikuje przedmiot operacji.

Tabela 2.6. *Powszechnie stosowane przedrostki*

Przedrostek	Komponent
`Cc`	Menedżer pamięci podręcznej
`Cm`	Menedżer konfiguracji
`Ex`	Procedury pomocnicze centrum wykonawczego
`FsRtl`	Funkcje biblioteczne sterownika systemu plików
`Hal`	Warstwa uniezależnienia od sprzętu
`Io`	Menedżer I/O
`Ke`	Jądro
`Lpc`	Lokalne wywołanie procedury
`Lsa`	Lokalna kontrola bezpieczeństwa
`Mm`	Menedżer pamięci
`Nt`	Usługi systemowe Windows 2000 (z których większość dostępnych jest za pośrednictwem funkcji Win32)
`Ob`	Menedżer obiektów
`Po`	Menedżer energii
`Pp`	Menedżer PnP
`Ps`	Obsługa procesów
`Rtl`	Funkcje biblioteczne
`Se`	Bezpieczeństwo
`Wmi`	Instrumentacja zarządzania Windows
`Zw`	Alternatywne punkty wejściowe do usług systemowych (rozpoczynających się od `Nt`), które ustawiają poprzedni tryb zaufania na tryb jądra. Eliminuje to sprawdzanie poprawności argumentów, ponieważ usługi systemowe `Nt` sprawdzają parametry tylko, jeśli poprzednim trybem zaufania był tryb użytkownika

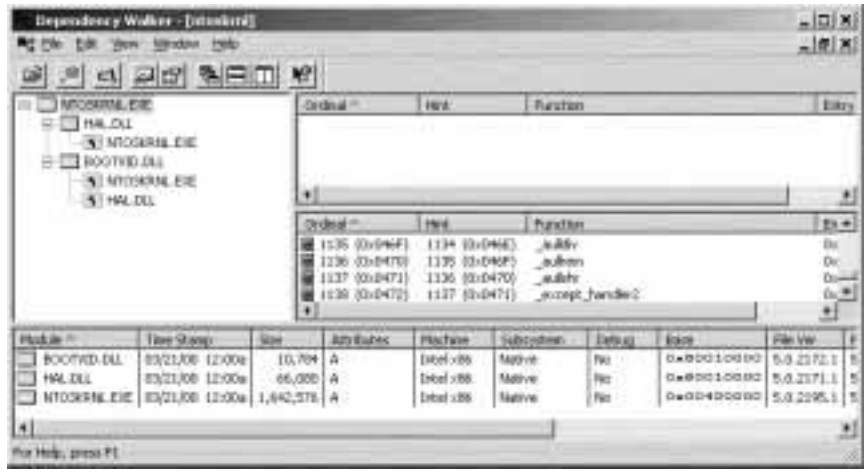
Na przykład `ExAllocatePoolWithTag` jest pomocniczą funkcją przydzielającą pamięć z puli stronicowanej lub niestronicowanej. `KeInitializeThread` jest procedurą, która tworzy i ustawia atrybuty obiektu reprezentującego wątek jądra.

ĆWICZENIE: Lista funkcji nieudokumentowanych

Istnieje możliwość obejrzenia tabel funkcji eksportowanych i importowanych przez plik binarny — służy do tego narzędzie *Dependency Walker* (*Depends.exe*), wchodzące w skład pakietów Windows 2000 Support Tools oraz Platform SDK. Aby zanalizować plik binarny narzędziem *Dependency Walker*, należy z menu *File* wybrać pozycję *Open* i wskazać interesujący plik.

Poniżej (patrz rysunek) znajduje się przykładowy rezultat działania tego programu — analizowane były zależności w pliku *Ntoskrnl*.

Warto zauważyć, że *Ntoskrnl* zawiera wywołania HAL, który z kolei korzysta z funkcji *Ntoskrnl*. (Korzystają nawzajem ze swoich procedur). *Ntoskrnl* jest również związany z biblioteką *Bootvid.dll*, rozruchowym sterownikiem wideo, wyświetlającym ekran startowy graficznego interfejsu użytkownika Windows 2000.



Szczegółowy opis danych wyświetlanych przez ten program znajduje się w jego pliku pomocy (*Depends.hlp*).

Procesy Systemowe

Poniższe procesy systemowe występują w każdym systemie Windows 2000. (Dwa z nich — jałowy i systemowy — nie są pełnymi procesami, gdyż nie wykonują one żadnego pliku binarnego w trybie użytkownika).

- Proces jałowy (zawiera po jednym wątku dla każdego procesora, który to wątek jest wykonywany w jałowym czasie procesora)
- Proces systemowy (zawiera większość wątków systemowych trybu jądra)
- Menedżer sesji (*Smss.exe*)
- Podsystem Win32 (*Csrss.exe*)
- Proces logowania (*Winlogon.exe*)
- Menedżer kontroli usług (*Services.exe*) oraz tworzone przez niego potomne procesy usługowe
- Lokalny serwer bezpieczeństwa (*Lsass.exe*)

Pomóc w zrozumieniu zależności pomiędzy tymi procesami pomóc może polecenie `tlist /t` z pakietu Windows 2000 Support Tools, które wyświetla „drzewo procesów", to znaczy powiązania typu rodzic-potomek pomiędzy procesami. Poniżej znajduje się skomentowany fragment rezultatu wykonania polecenia `tlist /t`:

```
System Process (0)          Proces jałowy
System (8)                  Proces systemowy (domyślnie zawiera wątki systemowe)
  smss.exe (144)            Menedżer sesji
    csrss.exe (172)         Proces podsystemu Win32
    winlogon.exe (192)      Proces logowania (zawiera również usługę NetDDE)
      services.exe (220)    Menedżer kontroli usług
        svchost.exe (384)   Kod serwera usług
        spoolsv.exe (480)   Usługa spoolera
        regsvc.exe (636)    Usługa zdalnego rejestru
        mstask.exe (664)    Usługa szeregowania zadań
      lsass.exe (232)       Lokalny serwer bezpieczeństwa
```

Kolejne podrozdziały zawierają opis kluczowych procesów systemowych zawartych na powyższym wydruku. Mimo że jest w nich zaznaczona kolejność uruchamiania procesów, rozdział 4. zawiera szczegółowy opis czynności koniecznych do rozruchu i startu Windows 2000.

Proces jałowy

Mimo podanej nazwy pierwszy proces na wydruku rezultatu polecenia `tlist /t` (identyfikator procesu 0) w rzeczywistości jest systemowym procesem jałowym. Jak to zostanie wyjaśnione w rozdziale 6., procesy są identyfikowane poprzez nazwę wykonywanego pliku binarnego. Jednak ten proces (podobnie jak proces 8, noszący nazwę *System*) nie wykonuje żadnego programu trybu użytkownika. Z tego powodu nazwy tych procesów wyświetlane przez różne programy narzędziowe różnią się. Mimo że większość narzędzi nazywa proces 8 „System", jednak nie wszystkie tak robią. Tabela 2.7 zawiera kilka nazw nadawanych procesowi jałowemu (o identyfikatorze 0). Proces jałowy jest omówiony szczegółowo w rozdziale 6.

Tabela 2.7. *Nazwy Procesu 0 w różnych programach narzędziowych*

Program narzędziowy	Nazwa procesu 0
Menedżer zadań	*Systemowy proces bezczynny*
Process Viewer (*Pviewer.exe*)	*Idle*
Process Status (*Pstat.exe*)	*Idle Process*
Process Explode (*Pview.exe*)	*System Process*
Task List (*Tlist.exe*)	*System Process*
QuickSlice (*Qslice.exe*)	*Systemprocess*

Pora teraz omówić wątki systemowe i zadania każdego z procesów systemowych wykonujących pliki binarne.

Proces systemowy i wątki systemowe

Proces systemowy (o identyfikatorze zawsze równym 8) zawiera wątki specjalnego rodzaju, wykonujące się jedynie w trybie jądra: *wątki systemowe trybu jądra*. Wątki systemowe mają wszystkie atrybuty i konteksty zwykłych wątków trybu użytkownika (takie jak kontekst procesora, priorytet, itd.), ale wyróżniają się tym, że działają jedynie w trybie

jądra, wykonując kod załadowany do przestrzeni jądra, czy to przez *Ntoskrnl.exe*, czy też dowolny załadowany sterownik urządzenia. Dodatkowo, wątki systemowe nie posiadają przestrzeni adresowej dostępnej w trybie użytkownika, przez co muszą zgłaszać żądania alokacji pamięci dynamicznej ze stert systemu operacyjnego(z puli pamięci stronicowanej lub niestronicowanej).

Wątki systemowe są tworzone przez funkcję `PsCreateSystemThread` (udokumentowaną w DDK), która może być wywołana jedynie z trybu jądra. Windows 2000, jak również różne sterowniki urządzeń, tworzą wątki systemowe podczas inicjalizacji systemu, celem przeprowadzenia operacji wymagających kontekstu wątku, jak żądanie lub oczekiwanie na operacje I/O lub inne obiekty czy polling urządzenia. Na przykład menedżer pamięci wykorzystuje wątki systemowe do implementacji takich funkcji, jak zapisywanie zmodyfikowanych stron do pliku wymiany lub plików odwzorowanych, przerzucanie procesów do i z pamięci, itd. Jądro tworzy wątek systemowy nazywany *menedżerem zbioru równowagi* który uaktywnia się co sekundę i może wyzwalać różne zdarzenia związane z szeregowaniem i zarządzaniem pamięcią. Menedżer pamięci podręcznej również korzysta z wątków systemowych do implementacji operacji wejścia-wyjścia z wyprzedzeniem odczytu i z opóźnieniem zapisu. Sterownik serwera plików (*Srv.sys*) korzysta z wątków systemowych do reagowania na sieciowe żądania danych z plików zwartych na partycjach udostępnionych w sieci. Nawet sterownik napędu dyskietek ma wątek systemowy wykonujący polling urządzenia (polling jest wydajniejszy w tym przypadku, ponieważ sterownik oparty na przerwaniach wymaga dużej ilości zasobów systemowych). Dalsze informacje o konkretnych wątkach systemowych są zawarte w rozdziałach opisujących dane komponenty.

Domyślnie, wątki systemowe wchodzą w skład procesu systemowego, jednak sterownik urządzenia może stworzyć wątek systemowy w dowolnym procesie. Na przykład sterownik podsystemu Win32 (*Win32k.sys*) tworzy wątki systemowe w procesie podsystemu Win32 (*Csrss.exe*), dzięki czemu mogą one łatwo odwoływać się do danych tego procesu, zawartych w przestrzeni adresowej trybu użytkownika

W przypadku lokalizacji błędów lub analizy systemu pożyteczna jest możliwość sprawdzenia sterownika czy nawet procedury zawierającej kod wykonywany w ramach konkretnego wątku. Na przykład w silnie obciążonym serwerze plików proces systemowy prawdopodobnie będzie zabierał znaczną część czasu procesora. Ale wiedza, że gdy aktywny jest proces systemowy, to aktywny jest „jakiś wątek systemowy", nie wystarcza do określenia, który sterownik lub komponent systemowy jest aktywny.

Tak więc, gdy aktywny jest proces systemowy, można obejrzeć wykonywanie wątków tego procesu (korzystając na przykład z narzędzia *Wydajność*). Po zlokalizowaniu aktywnego wątku (lub wątków) należy sprawdzić, w którym sterowniku rozpoczął działanie ten wątek systemowy (co jest informacją o tym, który sterownik prawdopodobnie utworzył wątek) lub zbadać stos wywołań (lub przynajmniej aktualny adres) badanego wątku, co oznaczałoby, gdzie aktualnie wykonywany jest wątek.

Obydwie techniki są zilustrowane w poniższych ćwiczeniach.

ĆWICZENIE: Identyfikacja wątków systemowych w procesie systemowym

Łatwo stwierdzić, że wątki wchodzące w skład procesu systemowego są wątkami systemowymi trybu jądra, ponieważ adres każdego wątku jest większy niż początkowy adres przestrzeni systemowej (która domyślnie rozpoczyna się od adresu 0x80000000, chyba że system uruchomiono z opcją `/3GB` w pliku *Boot.ini*). Również liczniki czasu procesora dla tych wątków wskazują, że wątki, które były aktywne przez jakiś czas, ten czas spędziły wyłącznie w trybie jądra.

Aby sprawdzić, który sterownik utworzył wątek systemowy, powinieneś zwrócić uwagę na adres startowy wątku (który można sprawdzić za pomocą programu *Pviewer.exe*), a następnie znaleźć sterownik, którego adres bazowy znajduje się najbliżej (ale przed) adresem startowym wątku. Zarówno narzędzie *Pstat* (pod koniec danych wyjściowych), jak i polecenie `!drivers` z programu uruchomieniowego jądra wyświetlają adresy bazowe załadowanych sterowników.

Aby szybko określić aktualny adres wątku, można skorzystać z polecenia `!stacks 0` programu uruchomieniowego jądra. Poniżej znajduje się przykładowy wynik tego polecenia w działającym systemie (przy wykorzystaniu *LiveKd*):

```
kd> !stacks 0
Proc.Thread  Thread   ThreadState  Blocker
                                   [System]
   8.000004  fe504da0  READY        ntoskrnl!MmZeroPageThread+0x5f
   8.00000c  fe5046a0  BLOCKED      ?? Kernel stack not resident ??
   8.000010  fe504420  BLOCKED      ntoskrnl!ExpWorkerThread+0x73
   8.000014  fe503020  BLOCKED      ntoskrnl!ExpWorkerThread+0x73
   8.000018  fe503da0  BLOCKED      ?? Kernel stack not resident ??
   8.00001c  fe503b20  BLOCKED      ntoskrnl!ExpWorkerThread+0x73
   8.000020  fe5038a0  BLOCKED      ?? Kernel stack not resident ??
   8.000024  fe503620  BLOCKED      ?? Kernel stack not resident ??
   8.000028  fe5033a0  BLOCKED      ntoskrnl!ExpWorkerThread+0x73
   8.00002c  fe502020  BLOCKED      ntoskrnl!ExpWorkerThread+0x73
   8.000030  fe502da0  BLOCKED      ntoskrnl!ExpWorkerThreadBalanceManager+0x55
   8.000034  fe5013a0  BLOCKED      ntoskrnl!MiDereferenceSegmentThread+0x44
   8.000038  fe501120  BLOCKED      ntoskrnl!MiModifiedPageWriterWorker+0x31
   8.00003c  fe500020  BLOCKED      ntoskrnl!KeBalanceSetManager+0x7e
   8.000040  fe500da0  BLOCKED      ntoskrnl!KeSwapProcessOrStack+0x24
   8.000044  fe500520  BLOCKED      ntoskrnl!FsRtlWorkerThread+0x33
   8.000048  fe4fc020  BLOCKED      ntoskrnl!FsRtlWorkerThread+0x33
   8.00004c  fe519a60  BLOCKED      ACPI+0xd8a6
   8.000050  fe505d00  BLOCKED      ntoskrnl!MiMappedPageWriter+0x4d
   8.000054  fe4b5da0  BLOCKED      dmio+0x7e91
   8.000058  fe4b1c80  BLOCKED      NDIS+0x52ac
   8.00005c  fe3d2da0  BLOCKED      raspptp+0x1b6d
   8.000064  fe3d2b20  BLOCKED      raspptp+0x1bc4
   8.000068  fe3c71e0  BLOCKED      rasacd+0xe66
   8.00006c  ff415da0  BLOCKED      rdbss!RxSetMinirdrCancelRoutine+0x585
   8.000070  ff415b20  BLOCKED      rdbss!RxSetMinirdrCancelRoutine+0x585
   8.000074  ff4158a0  BLOCKED      rdbss!RxSetMinirdrCancelRoutine+0x585
   8.000078  ff415620  BLOCKED      rdbss!RxIndicateChangeOfBufferingState+0xcdb
   8.00007c  ff411020  BLOCKED      ?? Kernel stack not resident ??
```

```
8.0001c8  ff3400e0  BLOCKED      ?? Kernel stack not resident ??
8.000258  ff33a0a0  BLOCKED      ?? Kernel stack not resident ??
8.000260  ff336da0  BLOCKED      ?? Kernel stack not resident ??
8.00023c  ff349b40  READY        parallel+0xb63c
```

Pierwsza kolumna zawiera identyfikatory procesu i wątku (w formacie „ID procesu.ID wątku"). Druga kolumna zawiera aktualny adres wątku. Trzecia kolumna określa, czy wątek znajduje się w stanie oczekiwania, gotowości czy aktywności. (Rozdział 6. zawiera opis możliwych stanów wątku). Ostatnia kolumna zawiera adres ze szczytu stosu wątku. Informacja w tej kolumnie ułatwia identyfikację sterownika, w którym rozpoczął się wątek. W przypadku wątków modułu *Ntoskrnl* nazwa funkcji zapewnia dodatkowe informacje na temat działania procesu.

Jednakże, jeśli badany wątek jest jednym z systemowych wątków wykonawczych (`ExpWorkerThread`), wciąż nie wiadomo, co takiego on robi, ponieważ każdy sterownik urządzenia może zlecić wątkowi wykonawczemu jakieś zadanie. Dlatego też jedynym sposobem wyśledzenia działalności wątku jest ustawienie pułapki na `ExQueueWorkItem`. Po zatrzymaniu programu na pułapce powinieneś wpisać polecenie `!dso work_queue_item esp+4`. To polecenie wyświetli pierwszy argument przekazany do `ExQueueWorkItem` (który jest strukturą kolejki roboczej), zawierający z kolei adres procedury wykonawczej, która zostanie wywołana w ramach wątku wykonawczego. Oprócz tego można sprawdzić, kto wywołuje procedurę. Sprawdzisz to, stosując w programie uruchomieniowym jądra polecenie `k`, które wyświetla aktualny stos wywołań. Aktualny stos wywołań zawiera informację o sterowniku, który zleca zadanie wątkowi wykonawczemu (w odróżnieniu od procedury, jaka ma zostać wywołana z tego wątku).

ĆWICZENIE: Identyfikacja sterownika związanego z wątkiem systemowym

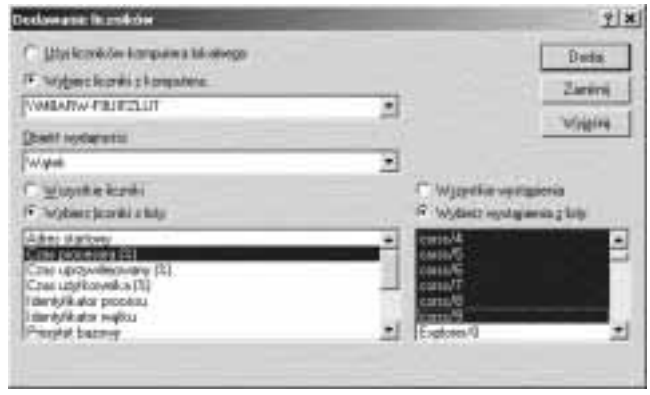
W tym ćwiczeniu zlokalizowany zostanie wątek niskopoziomowej obsługi myszy — wątek systemowy w sterowniku podsystemu Win32 (*Win32k.sys*) określający, które wątki powinny zostać powiadomione o ruchu myszy oraz zdarzeniach z nią związanych. Aby spowodować działanie tego wątku systemowego, wystarczy gwałtownie poruszać myszą, obserwując czas procesora dla procesów (za pomocą *Menedżera zadań*, narzędzia *Wydajność* lub narzędzia *QuickSlice* z Windows 2000 Resource Kit) — daje się zauważyć, że proces *Csrss* pracuje przez krótką chwilę.

Poniższe punkty pokazują, w jaki sposób zejść do poziomu wątków, by znaleźć sterownik zawierający aktywny wątek systemowy. Mimo że ten przykład demonstruje działanie wątku systemowego w ramach procesu *Csrss*, technikę tę można zastosować do identyfikacji sterownika, który utworzył wątek działający w ramach procesu systemowego.

Po pierwsze, musisz przygotować narzędzie *Wydajność* do obserwacji działania każdego wątku systemowego na *Monitorze systemu*. Aby to zrobić:

1. Uruchom narzędzie *Wydajność*.
2. Wybierz *Monitor systemu* i naciśnij przycisk *Dodaj* (znak plus na pasku narzędziowym).

3. Z listy *Obiekt wydajności* wybierz obiekt *Wątek*.
4. Wybierz licznik o nazwie *Czas procesora (%)* (lub *Czas uprzywilejowany (%)* — wartość będzie identyczna).
5. W liście *Wystąpień* przewiń kursor do wątku o nazwie *csrss/0*. Naciśnij tę pozycję, po czym przeciągnij myszą w dół, aby zaznaczyć wszystkie wątki procesu *csrss*. (W przypadku śledzenia wątków procesu systemowego należałoby wybrać wątki od *System/0* do ostatniego wątku procesu systemowego).
6. Naciśnij przycisk *Dodaj*.
7. Ekran programu powinien wyglądać podobnie do poniższego rysunku.



8. Naciśnij przycisk *Zamknij*.

Teraz należy zmienić skalowanie pionowej osi wykresu — z maksimum równego 100 na 10. Ułatwi to obserwowanie działania wątku systemowego, ponieważ wątki systemowe z reguły pozostają aktywne przez bardzo krótki czas. Aby zmienić skalowanie pionowej osi, powinieneś wykonać następujące czynności:

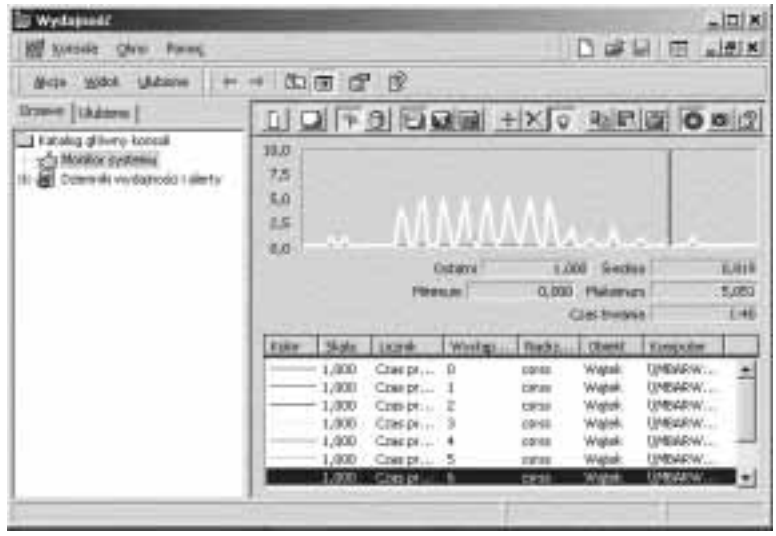
9. Naciśnij prawym przyciskiem myszy na wykresie.
10. Wybierz *Właściwości*.
11. Wybierz zakładkę *Wykres*.
12. Zmień wartość *Maksimum* w sekcji *skala pionowa* z 100 na 10.
13. Naciśnij przycisk *OK*.

Teraz, gdy śledzony jest proces *Csrss*, musisz spowodować jego działanie poprzez poruszanie myszą. Spowoduje to uaktywnienie wątków sterownika *Win32k.sys*. Następnie zidentyfikujemy sterownik, który utworzył proces systemowy aktywny podczas poruszania myszą:

14. Poruszaj gwałtownie myszą, aż zauważysz aktywność jednego lub dwóch wątków na wykresie narzędzia *Wydajność*.
15. Naciśnij *Ctrl+H*, aby włączyć tryb podświetlenia. (Powoduje to podświetlenie aktualnie zaznaczonego licznika na biało).

16. Przejrzyj liczniki celem identyfikacji wątku, który był aktywny podczas poruszania myszą (może być więcej niż jeden).

17. Zwróć uwagę na względny numer wątku w kolumnie Instancji, pod wykresem narzędzia *Wydajność* (wątek 6 w *csrss* na poniższym rysunku).



W tym przypadku aktywny był wątek 6 w procesie *csrss*. Teraz powinniśmy odnaleźć adres startowy tego wątku. Możemy to zrobić, stosując program *Process Viewer* z pakietu Windows 2000 Support Tools.

18. Uruchom program *Process Viewer*.

19. Naciśnij linię procesu *csrss* w liście wyświetlanych procesów.

20. Przewiń listę wątków w poszukiwaniu wątku o takim samym numerze względnym, jak ten otrzymany w punkcie 17.

21. Zaznacz ten wątek.

Jeśli znaleziony wątek należy do wątków systemowych procesu *csrss*, jego adres startowy (wyświetlany u dołu okna programu *Process Viewer* w sekcji *Thread Information*) powinien wynosić 0xa0009cbf. (Adres ten może być inny w nowszych wersjach Windows 2000).

22. I wreszcie, uruchom *Pstat.exe*, dostępny w pakiecie Platform SDK bądź na witrynie *www.reskit.com*.

Na końcu informacji generowanych przez *Pstat* znajduje się lista wszystkich załadowanych sterowników urządzeń, włącznie z ich adresami startowymi w systemowej pamięci wirtualnej. Poszukiwany jest sterownik o najbliższym adresie startowym, *poprzedzającym* adres startowy badanego wątku. Poniżej znajduje się fragment tego obszaru danych generowanych przez narzędzie *Pstat*:

```
ModuleName   Load Addr   Code    Data   Paged          LinkDate
-------------------------------------------------------------
ntoskrnl.exe 80400000  429184   96896  775360 Tue Dec 07 18:41:11 1999
     hal.dll 80062000   25856    6016   16160 Tue Nov 02 20:14:22 1999
 BOOTVID.DLL F7410000    5664    2464       0 Wed Nov 03 20:24:33 1999
    ACPI.sys BFFD8000   92096    8960   43488 Wed Nov 10 20:06:04 1999
  WMILIB.SYS F75C8000     512       0    1152 Sat Sep 25 14:36:47 1999
     pci.sys F7000000   12704    1536   31264 Wed Oct 27 19:11:08 1999
  isapnp.sys F7010000   14368     832   22944 Sat Oct 02 16:00:35 1999
...

  win32k.sys A0000000 1520960   54944       0 Tue Nov 30 03:51:03 1999
   rdbss.sys BEC8F000   27808    1952   86656 Tue Nov 30 03:52:29 1999

  mrxsmb.sys BEBF7000   91616   21824  237568 Tue Nov 30 03:52:10 1999
...
   ntdll.dll 77F80000  294912   12288   16384 Wed Oct 27 16:06:08 1999
-------------------------------------------------------------
       Total          4375648  547040 3164960
```

Łatwo zauważyć, że adres startowy badanego wątku, 0xa0009cbf, wchodzi w skład sterownika *Win32k.sys*, ponieważ nie istnieje sterownik o bliższym adresie startowym.

Jeśli adres znajduje się w obszarze zajmowanym przez *Ntoskrnl.exe*, oznacza to, że wątek wykonuje procedurę głównego modułu jądra. Sama ta informacja nie wystarczy do określenia, co robi wątek — trzeba znaleźć nazwę funkcji znajdującej się pod tym adresem. Można tego dokonać, przeglądając listę symboli globalnych, zawartą w tabeli symboli, w pliku *Ntoskrnl.dbg*.

Najprostszym sposobem wygenerowania listy symboli globalnych z *Ntoskrnl* jest uruchomienie programu uruchomieniowego jądra (przez połączenie z działającym systemem lub przez otwarcie pliku zrzutu awaryjnego) i wykonanie polecenia `x nt!*`, przy załadowanym jedynie pliku *Ntoskrnl.dbg*. Przed rozkazem *x* `nt!*` należy wydać polecenie `.logopen`, aby stworzyć dziennik sesji programu uruchomieniowego jądra. W ten sposób wyniki działania programu będą zapisywane w pliku, w którym łatwo znaleźć poszukiwane adresy. Można również skorzystać z narzędzia *Dumpbin* z pakietu Visual C++ (polecenie *dumpbin /symbols ntoskrnl.dbg*), ale wtedy należy poszukiwać adresu powstałego przez odjęcie adresu bazowego *Ntoskrnl*, ponieważ w liście zawarte są przesunięcia adresów.

Oprócz tego, do wyświetlenia nazwy sterownika oraz funkcji pod aktualnym adresem wątku można wykorzystać polecenie `!stacks 0` programu uruchomieniowego jądra (zakładając, że załadowany jest odpowiedni plik z symbolami).

Menedżer sesji (Smss)

Menedżer sesji (*\Winnt\System32\Smss.exe*) jest pierwszym procesem trybu użytkownika tworzonym w systemie. Właściwy proces *Smss* jest tworzony przez wątek systemowy trybu jądra, odpowiedzialny za ostatnią fazę inicjalizacji jądra i centrum wykonawczego.

Menedżer sesji jest odpowiedzialny za wiele istotnych czynności podczas uruchamiania systemu Windows 2000, jak otwarcie dodatkowych plików stronicowania, wykonanie opóźnionych operacji zmiany nazwy i usunięcia plików czy inicjalizacja systemowych

zmiennych środowiskowych. Uruchamia również procesy podsystemów (zwykle tylko *Csrss.exe*) oraz proces *Winlogon*, który z kolei tworzy pozostałe procesy w systemie.

Wiele z zawartych w rejestrze informacji konfiguracyjnych, które kierują działaniem procesu *Smss*, znajduje się pod adresem *HKLM\SYSTEM\CurrentControlSet\Control\Session Manager*. Analiza danych zawartych pod tym adresem może być interesująca. (Opis kluczy i wartości znajduje się w pliku pomocy dotyczącej pozycji rejestru, *Regentry.chm*, zawartym w Windows 2000 Resource Kit).

Po czynnościach inicjalizacyjnych główny wątek w *Smss* oczekuje w nieskończoność na uchwytach do *Csrss* i *Winlogon*. Jeśli którykolwiek z tych procesów w nieoczekiwany sposób zakończy się, *Smss* powoduje załamanie systemu, gdyż Windows 2000 opiera się na ich istnieniu. W międzyczasie, *Smss* oczekuje na żądania załadowania podsystemów, uruchamianie nowych podsystemów oraz zdarzenia związane z wykrywaniem błędów. Działa również jako przełącznik i monitor między aplikacjami oraz programami uruchomieniowymi.

Proces logowania (Winlogon)

Proces logowania Windows 2000 *\Winnt\System32\Winlogon.exe*) obsługuje interaktywne logowanie i wylogowywanie się użytkowników. *Winlogon* jest powiadamiany o żądaniu logowania użytkownika po naciśnięciu kombinacji klawiszy zwanej *bezpieczną sekwencją gotowości* (ang. *secure attention sequence — SAS*). Domyślną sekwencją SAS w Windows 2000 jest kombinacja *Ctrl+Alt+Delete*. Powodem istnienia kombinacji SAS jest ochrona użytkowników przed programami przechwytującymi hasła, symulującymi proces logowania. Po wprowadzeniu nazwy użytkownika i hasła są one wysyłane do procesu lokalnego serwera bezpieczeństwa (opisanego w następnym podrozdziale) celem sprawdzenia. Jeśli wszystko się zgadza, *Winlogon* pobiera wartość zmiennej rejestru o nazwie *Userinit*, spod klucza *HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon* i tworzy procesy wykonujące programy podane w tej zmiennej. Domyślnie uruchamiany jest proces o nazwie *Userinit.exe*.

Proces ten dokonuje częściowej inicjalizacji środowiska użytkownika (jak ustanowienie liter napędów mapowanych, wykonanie skryptu logowania i zastosowanie polityki bezpieczeństwa dla grupy), po czym odwołuje się do rejestru w poszukiwaniu wartości zmiennej `Shell` (we wspomnianym poprzednio kluczu *Winlogon*). Następnie tworzy proces, w którym uruchamiana jest powłoka systemu (domyślnie jest to *Explorer.exe*). Następnie *Userinit* kończy działanie. Z tego powodu *Explorer.exe* wyświetlany jest bez procesu macierzystego — jego proces macierzysty się zakończył, a jak to wspomniano wcześniej, program *Tlist* wyrównuje do lewej procesy, których rodzice nie istnieją. (Jeszcze inaczej można traktować *Explorer* jako wnuka procesu *Winlogon*).

Aspekty procesu logowania związane z uwierzytelnianiem i identyfikacją są zaimplementowane w wymienialnej bibliotece DLL o nazwie GINA (ang. *Graphical Identification and Authentication* — graficzna identyfikacja i uwierzytelnianie). Standardowa biblioteka GINA systemu Windows 2000, *Msgina.dll*, implementuje domyślny interfejs logowania Windows 2000. Jednakże projektanci mogą stworzyć własną wersję biblioteki GINA, implementującą inne mechanizmy identyfikacji i uwierzytelniania (na przykład oparte na rozpoznawaniu głosu), zamiast standardowej metody *nazwa/hasło*, stosowanej w Windows 2000. Oprócz tego *Winlogon* może załadować dodatkowe biblioteki dostawcy sieciowego,

dokonujące kolejnego uwierzytelniania. Zdolność ta pozwala wielu dostawcom sieciowym pobierać informacje dotyczące identyfikacji i uwierzytelniania podczas zwykłego logowania.

Winlogon uaktywnia się nie tylko podczas logowania i wylogowywania się użytkownika, ale także gdy przechwytuje sekwencję SAS z klawiatury. Na przykład naciśnięcie *Ctrl+Alt+Delete* przez zalogowanego użytkownika powoduje pojawienie się okna dialogowego systemu bezpieczeństwa Windows, które umożliwia wylogowanie się, uruchomienie *Menedżera zadań*, zablokowanie stacji roboczej, zamknięcie systemu, itd. To *Winlogon* jest procesem, który obsługuje te opcje.

Więcej informacji na temat procesu *Winlogon* znajduje się w rozdziale 8.

ĆWICZENIE: Obserwacja wielu sesji

Jeśli zainstalowane są usługi terminalowe, Menedżer sesji tworzy osobne instancje procesów *Csrss* oraz *Winlogon* dla każdego podłączonego użytkownika. Lista sesji może być wyświetlona za pomocą polecenia `!session` w programie uruchomieniowym jądra. W poniższym przykładzie w systemie znajdują się trzy aktywne sesje. (Polecenie `!session` pokazuje tylko proces *Csrss.exe* wchodzący w skład sesji, mimo że każda sesja zawiera więcej procesów).

```
kd> !session
**** NT ACTIVE SESSION DUMP ****
PROCESS 813531e0  Cid: 00c8    Peb: 7ffdf000  SessionId: 00000000
    DirBase: 03639000  ObjectTable: 81353508  TableSize: 371.
    Image: csrss.exe

PROCESS 81180c80  Cid: 0364    Peb: 7ffdf000  SessionId: 00000001
    DirBase: 00fcc000  ObjectTable: 812c7288  TableSize: 115.
    Image: csrss.exe

PROCESS 811358e0  Cid: 0618    Peb: 7ffdf000  SessionId: 00000002
    DirBase: 040f0000  ObjectTable: 8114bcc8  TableSize: 111.
    Image: csrss.exe
```

Lokalny serwer bezpieczeństwa (Local Security Authentication Server, LSASS)

Proces lokalnego serwera bezpieczeństwa (*\Winnt\System32\Lsass.exe*) otrzymuje żądania uwierzytelniania od procesu *Winlogon* i wywołuje odpowiedni pakiet uwierzytelniający (zaimplementowany w bibliotece DLL) celem dokonania właściwej weryfikacji, polegającej na przykład na sprawdzeniu, czy hasło zgadza się z wzorcem zapisanym w aktywnym katalogu lub bazie *SAM* (będącej częścią rejestru zawierającą definicje użytkowników i grup).

Po udanym uwierzytelnieniu *Lsass* generuje obiekt znacznik dostępu, zawierający profil bezpieczeństwa użytkownika. *Winlogon* korzysta z tego znacznika do stworzenia początkowego procesu powłoki systemowej. Procesy uruchamiane przez powłokę domyślnie dziedziczą ten znacznik dostępu.

Więcej szczegółów o *Lsass* i uwierzytelnianiu znajduje się w rozdziale 8. Opis dostępnych funkcji umożliwiających dostęp do *Lsass* (o nazwach rozpoczynających się od *Lsa*) znajduje się w dokumentacji Platform SDK.

Menedżer kontroli uslug (Service Control Manager, SCM)

Jak to już zostało wspomniane, termin „usługi" w Windows 2000 może odnosić się do procesów serwerowych lub do sterowników. Ta część zajmuje się usługami będącymi procesami trybu użytkownika. Usługi są podobne do „demonów" znanych z systemów Unix lub „procesów odłączonych" z systemu VMS. Wynika to z tego, że mogą być skonfigurowane tak, by uruchamiać się automatycznie podczas rozruchu systemu, nie wymagając interaktywnego logowania. Mogą być również uruchamiane ręcznie (za pośrednictwem narzędzia administracyjnego o nazwie Usługi lub poprzez wywołanie funkcji Win32 o nazwie `StartService`). Z reguły usługi nie wchodzą w interakcje z zalogowanym użytkownikiem, choć istnieją specjalne warunki, kiedy jest to możliwe (szczegóły w rozdziale 5).

Menedżer kontroli usług jest specjalnym procesem systemowym, którego kod jest zawarty w pliku *\Winnt\System32\Services.exe*, a który odpowiada za uruchamianie, zatrzymywanie i interakcje z procesami usługowymi. Programy usługowe są w rzeczywistości plikami wykonywalnymi Win32, które wywołują specjalne funkcje Win32, wchodzące w interakcje z menedżerem kontroli usług celem wykonania takich czynności, jak odnotowanie udanego rozruchu usługi, odpowiedzi na żądania dotyczące stanu czy pauza lub zatrzymanie usługi. Usługi są zdefiniowane w rejestrze, pod adresem *HKLM\SYSTEM\CurrentControlSet\Services*. Zawarty w Windows 2000 Resource Kit plik pomocy dotyczący pozycji rejestru (*Regentry.chm*) dokumentuje podklucze i wartości związane z usługami.

Warto pamiętać, że usługi posiadają trzy nazwy: nazwę procesu widoczną w systemie, wewnętrzną nazwę w rejestrze oraz nazwę wyświetlaną przez narzędzie administracyjne *Usług*. (Nie wszystkie usługi mają nazwę wyświetlaną — jeśli usługa nie posiada nazwy wyświetlanej, pokazywana jest nazwa wewnętrzna). W Windows 2000 usługi mogą również posiadać składające się z 1024 znaków pole opisu, zawierające dalsze szczegóły dotyczące działania usługi.

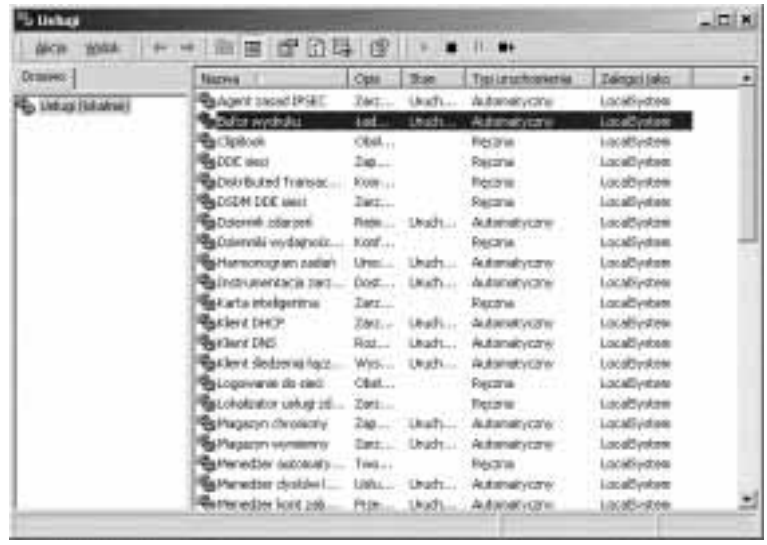
Aby przyporządkować proces usługowy do usług zawartych w tym procesie, powinieneś skorzystać z polecenia `tlist /s`. Warto zauważyć, że nie zawsze istnieje przyporządkowanie typu „jeden na jeden" między procesem usługowym a aktywnymi usługami, ponieważ kilka usług może wchodzić w skład jednego procesu. W rejestrze kod typu określa, czy usługa posiada swój własny proces, czy też współdzieli proces z innymi usługami zawartymi w pliku binarnym.

W postaci usług zostało zaimplementowanych wiele spośród komponentów Windows 2000, jak na przykład bufor wydruku, dziennik zdarzeń, mechanizm szeregowania zadań oraz różne komponenty sieciowe.

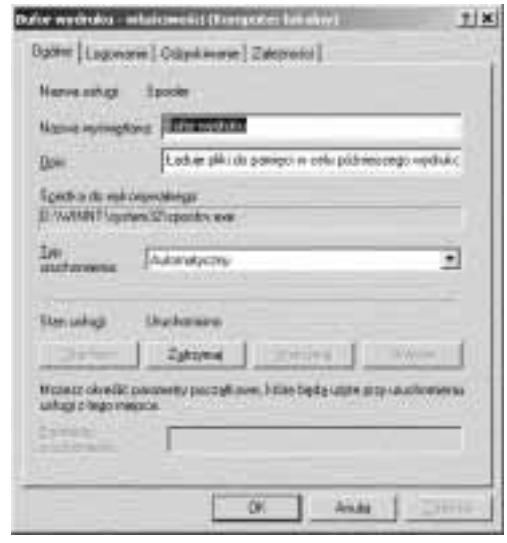
Więcej szczegółów na temat usług znajduje się w rozdziale 5.

ĆWICZENIE: Wyświetlanie listy zainstalowanych uslug

Aby wyświetlić listę zainstalowanych usług, z *Panelu sterowania* wybierz *Narzędzia administracyjne*, a następnie uruchom *Usługi*:



Aby uzyskać szczegółowe informacje o usłudze, na usłudze naciśnij prawy przycisk myszy i wybierz *Właściwości*. Na przykład poniżej znajdują się właściwości usługi *Bufora wydruku* (podświetlonej na poprzednim rysunku).



Warto zauważyć, że *Ścieżka do wykonywalnego* identyfikuje program zawierający tę usługę. Istotne jest, że niektóre usługi współdzielą proces z innymi usługami — odwzorowanie nie zawsze jest typu jeden-na-jeden.

Podsumowanie

W tym rozdziale została omówiona ogólnie architektura systemu Windows 2000. Przeanalizowane zostały kluczowe komponenty systemu oraz ich wzajemne zależności. W kolejnym rozdziale omówimy bardziej szczegółowo podstawowe mechanizmy systemowe, na których bazują te komponenty, jak menedżer obiektów oraz synchronizacja.